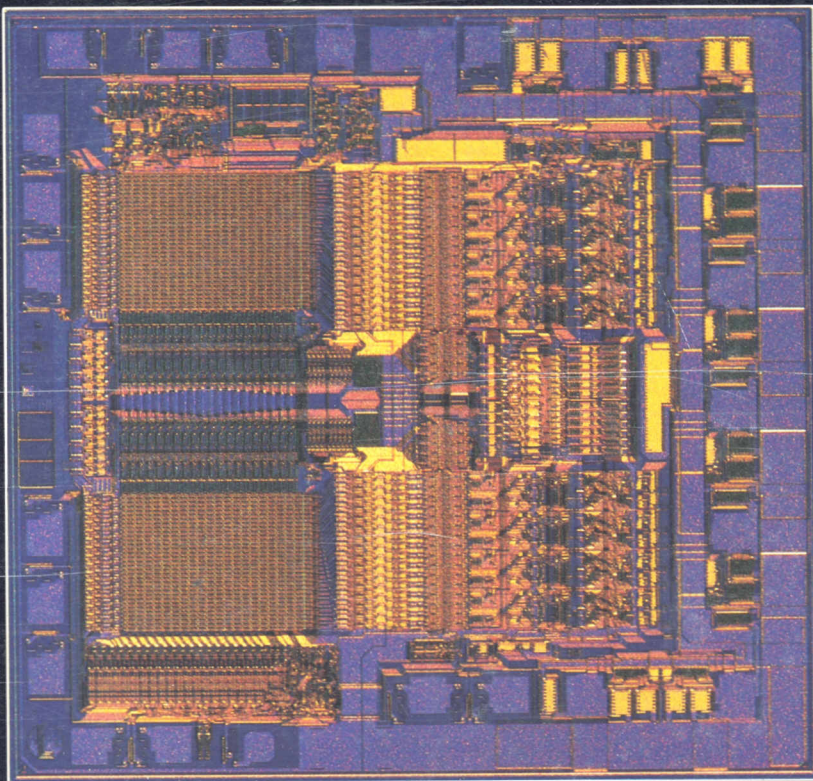


**GAL<sup>®</sup>**

GENERIC  
ARRAY LOGIC



**LATTICE**  
SEMICONDUCTOR CORP.

# GAL<sup>®</sup> HANDBOOK

## LATTICE



# GAL<sup>®</sup> HANDBOOK

© 1986 Lattice Semiconductor Corp.

GAL and UltraMOS are registered trademarks of Lattice Semiconductor Corporation. Generic Array Logic, RAL, Reprogrammable Array Logic, Latch-Lock and ComPALibility are trademarks of Lattice Semiconductor Corporation.

ABEL is a trademark of Data I/O Corporation; CUPL is a trademark of Assisted Technology Inc.; PAL, PALASM, and HAL are registered trademarks of Monolithic Memories, Inc.

The examples and applications contained in this publication are for illustration purposes only. Lattice Semiconductor assumes no responsibility for their suitability of use without further modification and testing. The Lattice Semiconductor Invoice/Acknowledgment contains the full Terms and Conditions of Sale of Lattice Products. Lattice Semiconductor makes no warranty, express, statutory, or implied regarding the information in this publication, or that the described devices are free from patent infringement. No licenses are granted or implied. Lattice Semiconductor assumes no responsibility and makes no warranty that the devices are suitable for any given application. The obligation with respect to claims shall be limited to repair or replacement of the nonconforming or defective product. Lattice Semiconductor reserves the right to change product availability, prices and specifications without notice at any time.

Lattice Semiconductor products are intended for usage in normal commercial applications. Use of these devices in applications requiring extended temperature or voltage operation, harsh environmental conditions, or high reliability applications such as military, industrial or medical systems is not recommended without Lattice's further processing, monitor, test and acceptance for those applications. Use of these devices in life-support and life-sustaining applications is neither recommended nor authorized without written permission from the President of Lattice Semiconductor.

<b>1 INTRODUCTION</b> .....	1-1	<b>6 TECHNICAL BRIEFS</b> .....	6-1
<b>2 GAL DEVICE SPECIFICATIONS</b> .....	2-1	Copying PAL Patterns into GAL Devices . . .	6-3
GAL16V8 .....	2-3	Generic Architecture — Anything, Everytime .....	6-8
GAL20V8 .....	2-17	Using Electronic Signature .....	6-16
GAL16V8-15 .....	2-31	Overview of Register Preload .....	6-19
ispGAL16Z8 .....	2-33	Testability Advantages of GAL Devices . . .	6-21
GAL39V18 .....	2-35	E <sup>2</sup> CMOS and Programmable Logic .....	6-24
HI-REL (X) GAL16V8 .....	2-39	GAL Device Power Considerations .....	6-26
HI-REL (X) GAL20V8 .....	2-53	Latch-Lock Eliminates Latch-up .....	6-30
<b>3 LOGIC TUTORIAL</b> .....	3-1	Hidden Costs in PLD Usage .....	6-33
Logic Design Fundamentals .....	3-2	<b>7 E<sup>2</sup>CMOS TECHNOLOGY OVERVIEW</b> .....	7-1
Programmable Logic Device Alternatives . .	3-6	E <sup>2</sup> CMOS Process Technology and Circuit Characteristics .....	7-3
<b>4 USING DEVELOPMENT TOOLS</b> .....	4-1	E <sup>2</sup> CMOS Cell Device Physics .....	7-10
Hardware and Software Tools .....	4-3	<b>8 GAL DEVICE QUALITY AND RELIABILITY</b> .8-1	
The Design Process .....	4-6	The Lattice Quality Assurance Program . . .	8-2
Design Example: A Two-Story Elevator . . .	4-9	GAL Device Reliability Data .....	8-4
<b>5 GAL DEVICE APPLICATIONS</b> .....	5-1	HI-REL (X) Processing and Product Flow . .8-6	
Introduction .....	5-2	MIL-STD-883C Processing and Product Flow .....	8-7
Basic Gates .....	5-3	<b>9 ARTICLE REPRINTS</b> .....	9-1
Basic Flip Flops .....	5-10	EEPROM Technology Seeds Reprogrammable Logic .....	9-3
Shift Register .....	5-15	A 16ns CMOS EEPLA with Reprogrammable Architecture .....	9-7
4-bit Up/Down Counter .....	5-19	Electrically Erasable Devices Provide Optimal Programmable Logic Device Quality . . .	9-13
7-bit Counter with Load and Carry .....	5-23	Design Methodology For New Programmable Logic Devices .....	9-23
Memory Address Decoder .....	5-28	E <sup>2</sup> CMOS Programmable Logic: Superior Quality, Flexibility, and System Performance .....	9-31
Barrel Shifter .....	5-32	<b>10 APPENDICES</b> .....	10-1
Quad 4:1 Multiplexer .....	5-39	GAL Device Icons .....	10-2
8:3 Priority Encoder .....	5-43	Package Outline Drawings .....	10-7
Password Decoder .....	5-48	Development Tool Support .....	10-12
Decoder with Wait-State Generator .....	5-52	Ordering Information .....	10-14
Bus Arbiter .....	5-55	<b>11 SALES OFFICES</b> .....	11-1
4-Bit Cascadable Adder .....	5-59		
Clock Stretcher .....	5-61		
Dual-Port DRAM Controller .....	5-65		
Three-Story Elevator .....	5-75		



1	INTRODUCTION	1-1
2	GAL DEVICE SPECIFICATIONS	2-1
	GAL16V8	2-2
	GAL16V8	2-17
	GAL16V8-16	2-21
	ispGAL16V8	2-22
	GAL20V10	2-22
	Hi-REL (X) GAL16V8	2-22
	Hi-REL (X) GAL20V10	2-22
3	LOGIC TUTORIAL	3-1
	Logic Design Fundamentals	3-2
	Programmable Logic Device Alternatives	3-4
4	USING DEVELOPMENT TOOLS	4-1
	Hardware and Software Tools	4-2
	The Design Process	4-6
	Design Example: A Two-Busy Elevator	4-8
5	GAL DEVICE APPLICATIONS	5-1
	Introduction	5-2
	Basic Gates	5-2
	Basic Flip Flops	5-10
	Shift Register	5-12
	4-bit Up/Down Counter	5-18
	7-bit Counter with Load and Carry	5-22
	Memory Address Decoder	5-28
	Barrel Shifter	5-32
	Quad 4:1 Multiplexer	5-32
	8:3 Priority Encoder	5-32
	Parity Decoder	5-48
	Decoder with Wall-Slice Generator	5-52
	Bus Arbitrator	5-52
	4-Bit Cascadable Adder	5-58
	Clock Stretcher	5-61
	Dual-Port DRAM Controller	5-65
	Three-Busy Elevator	5-75
6	TECHNICAL BRIEFS	6-1
	Copying PAL Patterns into GAL Devices	6-2
	Generic Architecture — Anything	6-9
	Everything	6-12
	Using Electronic Signature	6-12
	Overview of Register Pinout	6-12
	Testability Advantages of GAL Devices	6-21
	E <sup>2</sup> CMOS and Programmable Logic	6-24
	GAL Device Power Considerations	6-28
	Latch Lock Eliminates Latchup	6-30
	Hidden Costs in PLD Usage	6-30
7	E <sup>2</sup> CMOS TECHNOLOGY OVERVIEW	7-1
	E <sup>2</sup> CMOS Process Technology and Circuit Characteristics	7-2
	E <sup>2</sup> CMOS Cell Device Physics	7-10
8	GAL DEVICE QUALITY AND RELIABILITY	8-1
	The Lattice Quality Assurance Program	8-2
	GAL Device Reliability Data	8-4
	Hi-REL (X) Processing and Product Flow	8-6
	Product Flow	8-7
9	ARTICLE REPRINTS	9-1
	EEPROM Technology Seeds Reprogrammable Logic	9-2
	A True CMOS EEPROM with Reprogrammable Architecture	9-7
	Electrically Erasable Devices Provide Optimal Programmable Logic Device Quality	9-12
	Design Methodology for New Programmable Logic Devices	9-22
	E <sup>2</sup> CMOS Programmable Logic Superior Quality, Reliability and System Performance	9-24
10	APPENDICES	10-1
	GAL Device Icons	10-2
	Package Outline Drawings	10-7
	Development Tool Support	10-12
	Ordering Information	10-14
11	SALES OFFICES	11-1

#### Acknowledgements

To the many Lattice employees whose long hours and hard work helped make this book possible, we extend our gratitude. To the core team of authors and editors — Angela Brandt, R.P. Capece, Jerry Greiner, David Lee Rutledge, and Dean Suhr — our special thanks.

INTRODUCTION	1
GAL DEVICE SPECIFICATIONS	2
LOGIC TUTORIAL	3
USING DEVELOPMENT TOOLS	4
GAL DEVICE APPLICATIONS	5
TECHNICAL BRIEFS	6
E <sup>2</sup> CMOS TECHNOLOGY OVERVIEW	7
GAL DEVICE QUALITY AND RELIABILITY	8
ARTICLE REPRINTS	9
APPENDICES	10
SALES OFFICES	11





**Lattice Semiconductor's corporate headquarters,  
a 120,000-square-foot complex located a few miles  
west of Portland, Oregon.**

### INTRODUCTION

Lattice Semiconductor was founded in 1983 to design, develop and manufacture high-performance semiconductor components. It is a firm belief at Lattice that technological evolution can be accelerated through the continued development of higher-speed and architecturally superior products. This belief led to a decision to enter the programmable logic marketplace by developing the ideal product line: the GAL (Generic Array Logic) family of devices.

GAL devices are ideal for four important reasons:

1. GAL devices are fabricated using very high-speed Electrically Erasable CMOS (E<sup>2</sup>CMOS), which offers the highest degree of testability and quality of any process technology, as well as instant erasability, making GAL devices ideal for prototyping.
2. GAL device speeds are at least as fast as any other TTL-compatible programmable logic device available.
3. GAL devices have the low power consumption of CMOS.
4. GAL devices utilize Output Logic Macrocells (OLMCs), which allow the user to configure outputs as needed.

By melding all these features into a single product line, the GAL family is ideally targeted to replace TTL/74HC random logic, low-density gate arrays, and all other programmable logic. The GAL family offers the benefits of reduced system cost, product size, and power requirements, as well as higher reliability and greatly simplified system design.

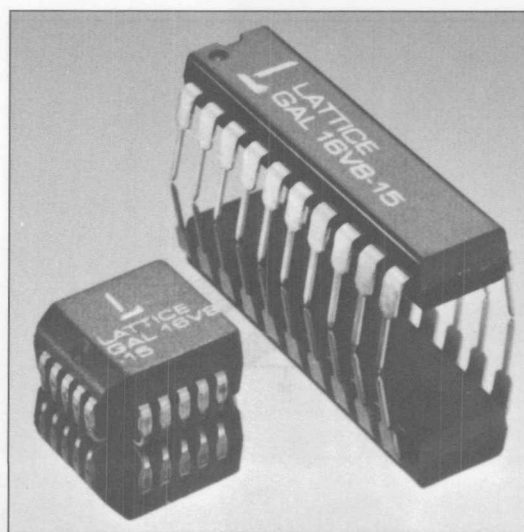
Lattice GAL devices outperform bipolar PLDs with the introduction of the ultrafast 15ns GAL16V8. With a maximum  $I_{CC}$  of 90mA, the GAL16V8-15 offers the best speed-power product in the industry.

### Why Use Programmable Logic?

As system design methodology continues to evolve, there becomes an increasing need to not only simplify the design process, but to reduce the overall system size and cost, and increase system reliability. It was this mindset that led to the development of the first programmable logic devices. In fact, the evolution of programmable logic has changed the way systems are designed, because it offers the designer a single tool that solves all his needs. Programmable logic is ideal for simplifying the design process, because the designer can implement the exact logic functions wherever and whenever required. It is also ideal for reducing system size and cost by offering significantly higher functional density than its small- and medium-scale (SSI/MSI) predecessors. Finally, system reliability is significantly improved because of simplified designs and lower parts count.

### Simplified System Design

Programmable logic is ideal as a design tool because the user specifies exactly what function or functions the devices will perform. The SSI/MSI approach, while capable of building up the same function, requires mixing and matching and interconnecting predefined chip functionality to arrive at a desired result. This is often a cumbersome process, resulting in underutilization of many of the SSI/MSI chips, as well as posing some significant board-layout problems. Programmable logic instead offers user-definable functionality that can be optimally tailored to any application. More efficient utilization, as well as reduced chip count, will greatly simplify the board-layout process, thereby streamlining the design process at both the conceptual stage and implementation stage.





### Increased Functional Density

Higher functional density than its SSI/MSI predecessors also contributes to making programmable logic an ideal design tool for reducing system size and cost. Functional density is the amount of logical functionality that can be compacted into a given space. Programmable logic can typically replace between four and twelve SSI/MSI packages performing the identical function. Also, since the user programs the function into the devices, functional utilization will be much higher than when dealing with the predefined functionality of SSI/MSI. These factors directly contribute to lower chip count, smaller boards, and reduced number of boards, resulting in an overall reduction in system size. It naturally follows that fewer chips and boards and smaller systems leads to significantly reduced development costs and, more importantly, manufacturing costs.

### Higher Reliability

Reduced chip count, reduced board count, and smaller system size contribute to much more than just lower cost. They also increase system reliability substantially. It has been statistically demonstrated that systems with higher levels of integration, such as those designed with programmable logic, have much higher reliability than equivalent systems designed with many low-density standard components. Simply put, there is less that can go wrong: fewer components to potentially fail, less interconnect on board, smaller, cooler-running, easier-to-manufacture systems, and so on, all contribute to higher reliability and are all direct benefits of designing with programmable logic.

The systems designer would like his task to be as simple as possible. The system user simply requires the

lowest cost, highest-reliability system available. Programmable logic provides these solutions, making it the ideal choice for design challenges.

## THE GAL CONCEPT

### E<sup>2</sup>CMOS — The Ideal Technology

Of the three major technology approaches available, E<sup>2</sup>CMOS, UVCMOS, and bipolar, the technology of choice is clearly E<sup>2</sup>CMOS — for many reasons, including: testability, quality, high speed, low power, and instant erasure for prototyping and error recovery.

### Testability

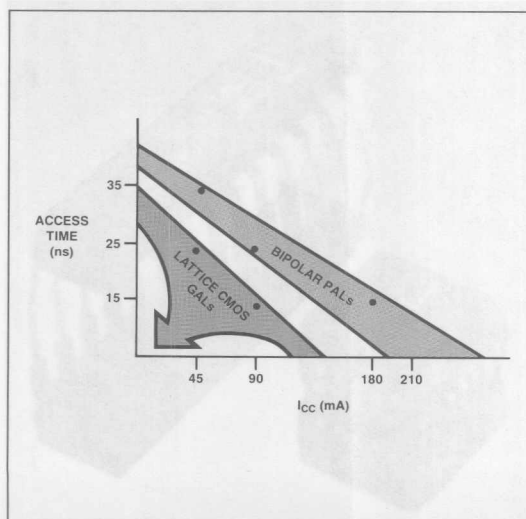
E<sup>2</sup>CMOS's biggest advantage over competing technologies is its inherent testability. Capitalizing on very fast (50ms) erase times allows Lattice to pattern and erase all devices many times during manufacture, to directly test all characteristics including AC, DC, and functionality. The result is guaranteed 100% programming and functional yields to the customer — and no further board rework. Competing technologies suffer serious test constraints, as discussed below.

### Low Power

Another advantage of this technology is the low power consumption of CMOS. This provides users the immediate benefit of decreased system power requirements allowing for higher-reliability, cooler-running systems, while maintaining high performance. The low power consumption of CMOS also permits circuit designs of much higher functional density, because of lower junction temperatures and power requirements on chip. The user will benefit because higher functional density means further reduction of chip count and smaller boards in the system.

### High Speed

Also advantageous is the very high speed attainable with Lattice's state-of-the-art E<sup>2</sup>CMOS process — speeds that are at least as fast as any device using any technology, with the exception of ECL circuits.



Using high-performance E<sup>2</sup>CMOS technology, Lattice GAL devices (colored band) match bipolar PLD speeds (grey band), while consuming half the power.

### Prototyping and Error Recovery

Finally, E<sup>2</sup>CMOS gives the user instant erasability, with no additional handling, or special packages necessary. This provides ideal products for prototyping because designs can be altered instantly, with no waste and no waiting. On the manufacturing floor, instant erasability can also be a big advantage for dealing with pattern changes or error recovery. If a GAL device is accidentally programmed to the wrong pattern, the recovery process is simple, again with no waiting or waste. Parts are simply put back into a device programmer and reprogrammed. No other technology can offer this.

### A Look at Other Technologies

Here, the technologies that compete with E<sup>2</sup>CMOS — bipolar and UVC MOS — are compared and contrasted with the E<sup>2</sup>CMOS approach.

#### Bipolar

Bipolar fuse-link technology was the first available for programmable logic devices. Although it offers high speed, it is saddled with high power dissipation. This not only significantly increases system power supply and cooling requirements but also limits the ability of high functional density.

Another shortcoming of this technology is the one-time-programmable fuses. Complete testing is impossible, and manufacturers must rely on complex schemes using test rows and columns to simulate and correlate their device's performance, since the fuse array cannot be tested prior to programming. The result is programming failures at the customer location, due to incomplete testing. Also, because these devices can only be programmed once, no reuse in the event of mistakes during prototyping or errors on the production floor are possible, and any misprogrammed devices must be discarded.

#### UVC MOS

UVC MOS addresses many of the shortcomings of the bipolar approach, but introduces many shortcomings of its own. This technology requires much lower power and, while it has the capability to erase, this comes at the expense of slower speeds and cumbersome erase procedures.

Testability is increased over bipolar since the 'fuse' array can be programmed by the manufacturer, to directly

test the functionality and performance. The problem here is the long (20 minutes) erase times of this technology, coupled with the requirement of exposing the devices to ultraviolet light for erasing. This becomes a very expensive step in the manufacturing process. Because of the time involved, patterning and erasing is performed only once — a compromised, rather than complete functional test.

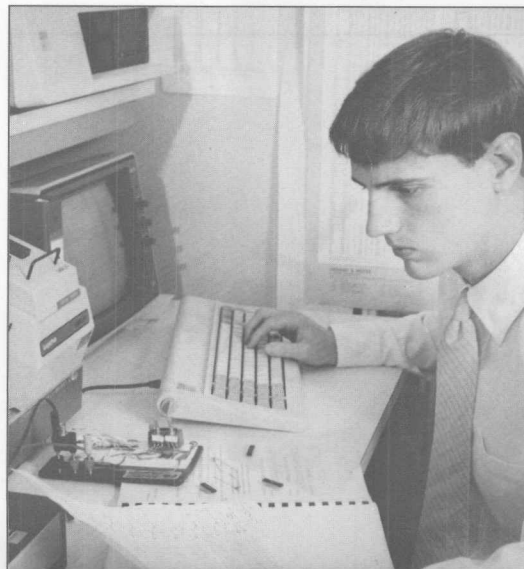
Additionally, the devices must be housed in expensive windowed packages to allow users to erase them. Again, this erase is coupled with the time-consuming and cumbersome task of shining ultraviolet light on the parts to erase them. As a cost-cutting measure, UVC MOS PLD manufacturers offer their devices in windowless packages, which cannot be completely tested after packaging, since they cannot be erased. Of course, the user cannot erase them either. These factors significantly detract from the desirability of this technology.

### The GAL Advantage

GAL devices are ideal programmable logic devices because, as the name implies, they are architecturally generic. Lattice has employed the macrocell approach, which allows users to define the architecture and functionality of each output. The key benefit to the user is the freedom from being tied to any specific functionality. This is advantageous at the manufacturing level, as well as at the design level.

### Design Advantages

Early programmable logic devices gave the user the ability to specify a function, but limited him to specific,



GAL devices are programmed using industry-standard hardware and software tools. A wide variety of third-party vendors now supports the Lattice GAL family.



predetermined output architectures. Comparing the GAL device with fixed-architecture programmable logic devices is much like comparing these same fixed PLDs with SSI/MSI. The GAL family is the next generation in simplified system design. The user needs not bother searching for the architecture that best suits a particular design. Instead, the GAL family's generic architecture lets him configure as he goes.

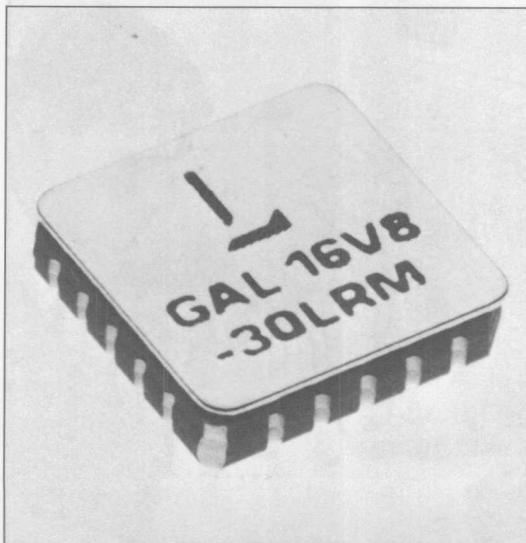
### Manufacturing Advantages

The one-device-does-all approach greatly simplifies manufacturing flow. Inventorying one generic-architecture GAL device type versus having to monitor and maintain many different device types, each with its own architecture, will not only save money, but will minimize the paperwork and headaches associated with the latter approach. Manufacturing flow is much smoother, too, because the handling process is greatly simplified. A generic-architecture GAL device also reduces the risk of running out of inventory and halting production, which can be a very expensive nightmare. Reduced chance of obsolete inventory and also easier QA tracking are additional benefits of the generic architecture.

### GAL Development Tools

Lattice Semiconductor supports the philosophy that users should not be required to purchase special development tools to use our products. As such, GAL devices are fully supported by existing industry-standard development tools. The tools required for designing with GAL products can be separated into two categories:

1. Programmable logic development software
2. Device programmers



### Development Software

Lattice Semiconductor believes the use of third-party development software is essential. Many device manufacturers offer their own software-development packages that support only their own devices. The disadvantage of this approach is that the user is required to purchase and maintain many new software packages every time each manufacturer introduces a new device. Third-party software packages, such as ABEL from Data I/O and CUPL from Assisted Technology, offer generic development support for all programmable logic devices, and, with periodic updates, the user will be kept up on all programmable logic device developments from all manufacturers. For these reasons, Lattice does not offer its own development software, but continues to work very closely with third-party vendors to ensure that the highest-quality development software is always available to support GAL devices.

### Device Programmers

As with development software, Lattice believes the use of third-party device programmers is essential. For the same reasons outlined in the software support program above, Lattice does not offer its own device programmers. Again, there is continued close work with third-party vendors to ensure that the highest-quality device programmers are available for programming GAL devices.

### The Ideal Package

Programmable logic devices are ideal for designing today's systems. Lattice Semiconductor believes that the ideal design approach should be supported with the ideal products. It was on this premise that GAL devices were invented. The ideal device — with a generic architecture — fabricated with the ideal process technology, E<sup>2</sup>CMOS.

We will continue to develop and expand our line of E<sup>2</sup>CMOS programmable logic devices, bringing higher speeds, more flexibility, and exciting new capabilities such as in-system programmability with our ispGAL family. This is the Lattice commitment to programmable logic and to you, our customer. □

Military GAL devices, fully tested over the  $-55^{\circ}$  to  $+125^{\circ}\text{C}$  temperature range, meet the needs of high-reliability applications in extreme environments.

INTRODUCTION	1
GAL DEVICE SPECIFICATIONS	2
LOGIC TUTORIAL	3
USING DEVELOPMENT TOOLS	4
GAL DEVICE APPLICATIONS	5
TECHNICAL BRIEFS	6
E <sup>2</sup> CMOS TECHNOLOGY OVERVIEW	7
GAL DEVICE QUALITY AND RELIABILITY	8
ARTICLE REPRINTS	9
APPENDICES	10
SALES OFFICES	11

INTRODUCTION	1
1.1 DEVICE SPECIFICATIONS	1
1.2 DEVICE TUTORIAL	1
2. USING DEVELOPMENT TOOLS	2
2.1 DEVICE APPLICATIONS	2
2.2 TECHNICAL BRIEFS	2
3. CMOS TECHNOLOGY OVERVIEW	3
3.1 QUALITY AND RELIABILITY	3
3.2 APPLICATIONS	3
APPENDICES	4
BIBLIOGRAPHY	4



## FEATURES

- **ELECTRICALLY ERASABLE CELL TECHNOLOGY**
  - Reconfigurable Logic
  - Reprogrammable Cells
  - Guaranteed 100% Yields
- **HIGH PERFORMANCE E<sup>2</sup>CMOS TECHNOLOGY**
  - Low Power: 45 mA Max Active  
35 mA Max Standby
  - High Speed: 25 ns Access Max  
35 ns Access Max
- **EIGHT OUTPUT LOGIC MACROCELLS**
  - Maximum Flexibility for Complex Logic Designs
  - Also Emulates 20-pin PAL<sup>®</sup> Devices with Full Function/Fuse Map/Parametric Compatibility
- **PRELOAD AND POWER-ON RESET OF ALL REGISTERS**
  - 100% Functional Testability
- **HIGH SPEED PROGRAMMING ALGORITHM**
- **SECURITY CELL PREVENTS COPYING LOGIC**
- **DATA RETENTION EXCEEDS 20 YEARS**

## DESCRIPTION

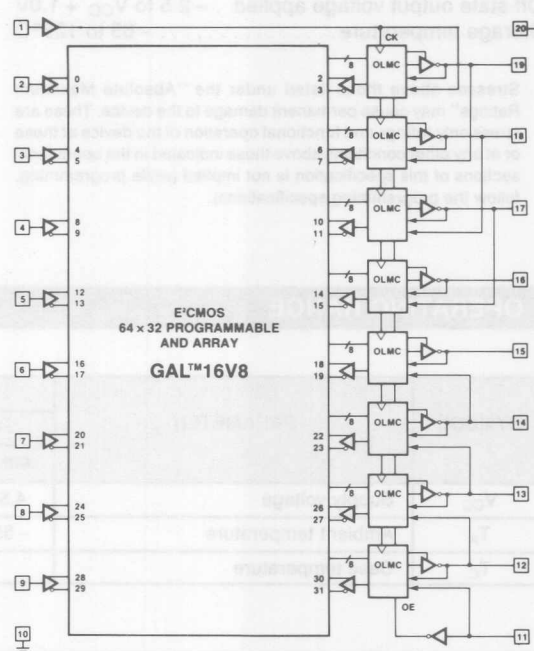
The LATTICE E<sup>2</sup>CMOS GAL device combines a high performance CMOS process with electrically erasable floating gate technology. This programmable memory technology applied to array logic provides designers with reconfigurable logic and bipolar performance at significantly reduced power levels.

The 20-pin GAL16V8 features 8 programmable Output Logic Macrocells (OLMCs) allowing each output to be configured by the user. Additionally, the GAL16V8 is capable of emulating, in a functional/fuse map/parametric compatible device, all common 20-pin PAL device architectures.

Programming is accomplished using readily available hardware and software tools. LATTICE guarantees 100 erase/write cycles and data retention exceeds 20 years.

Unique test circuitry and reprogrammable cells allow complete AC, DC, cell and functionality testing during manufacture. Therefore, LATTICE guarantees 100% field programmability and functionality of the GAL devices. In addition, electronic signature is available to provide positive device ID. A security circuit is built-in, providing proprietary designs with copy protection.

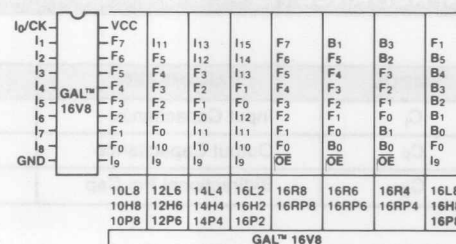
## FUNCTIONAL BLOCK DIAGRAM



## PIN NAMES

I <sub>0</sub> -I <sub>15</sub>	INPUT	$\overline{OE}$	OUTPUT ENABLE
CK	CLOCK INPUT	V <sub>CC</sub>	POWER (+ 5V)
B <sub>0</sub> -B <sub>5</sub>	BIDIRECTIONAL	GND	GROUND
F <sub>0</sub> -F <sub>7</sub>	OUTPUT		

## GAL16V8 EMULATING PAL DEVICES

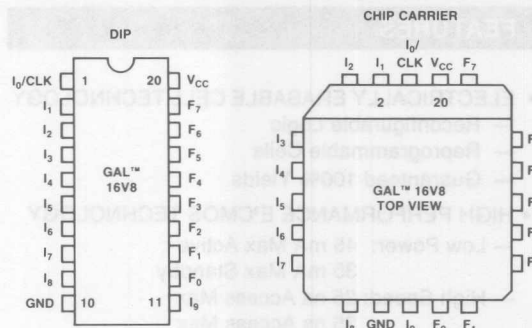


## ABSOLUTE MAXIMUM RATINGS<sup>(1)</sup>

Supply voltage  $V_{CC}$  ..... - .5 to +7V  
 Input voltage applied ..... - 2.5 to  $V_{CC} + 1.0V$   
 Off-state output voltage applied ..... - 2.5 to  $V_{CC} + 1.0V$   
 Storage temperature ..... - 65 to 125°C

- Stresses above those listed under the "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress only ratings and functional operation of the device at these or at any other conditions above those indicated in the operational sections of this specification is not implied (while programming, follow the programming specifications).

## PIN CONFIGURATION



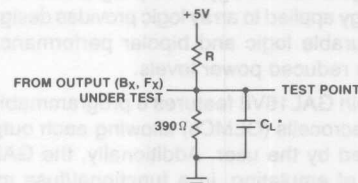
## OPERATING RANGE

SYMBOL	PARAMETER	TEMPERATURE RANGE						UNIT
		MILITARY			COMMERCIAL			
		MIN	TYP	MAX	MIN	TYP	MAX	
V <sub>CC</sub>	Supply voltage	4.5	5	5.5	4.75	5	5.25	V
T <sub>A</sub>	Ambient temperature	− 55			0		75	°C
T <sub>C</sub>	Case temperature			125				°C

## SWITCHING TEST CONDITIONS

Input Pulse Levels	GND to 3.0V
Input Rise and Fall Times	5ns 10% - 90%
Input Timing Reference Levels	1.5V
Output Timing Reference Levels	1.5V
Output Load	See Figure

3-state levels are measured 0.5V from steady-state active level.

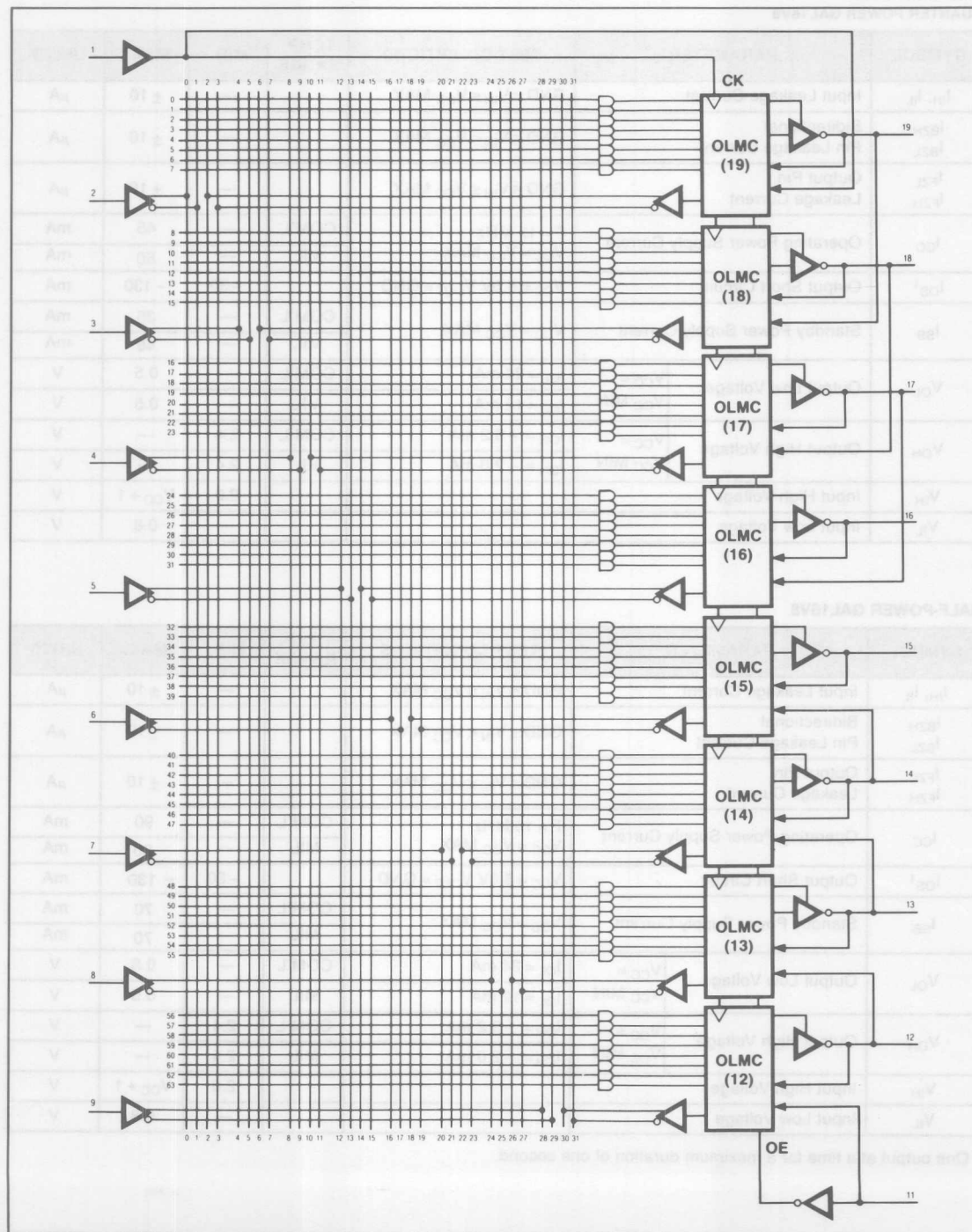


\* $C_L$  INCLUDES JIG AND PROBE TOTAL CAPACITANCE

## CAPACITANCE ( $T_A = 25^\circ\text{C}$ , $f = 1.0\text{ MHz}$ )

SYMBOL	PARAMETER	MAXIMUM	UNITS	TEST CONDITIONS
$C_I$	Input Capacitance	12	pF	$V_{CC} = 5.0V$ , $V_I = 2.0V$
$C_F$	Output Capacitance	15	pF	$V_{CC} = 5.0V$ , $V_F = 2.0V$
$C_B$	Bidirectional Pin Cap	15	pF	$V_{CC} = 5.0V$ , $V_B = 2.0V$

## GAL16V8 LOGIC DIAGRAM



**ELECTRICAL CHARACTERISTICS OVER OPERATING CONDITIONS**
**PRELIMINARY**
**QUARTER POWER GAL16V8**

SYMBOL	PARAMETER	TEST CONDITIONS	TEMP. RANGE	MIN.	MAX.	UNITS
$I_{IH}, I_{IL}$	Input Leakage Current	$GND \leq V_{IN} \leq V_{CC} \text{ MAX}$		—	$\pm 10$	$\mu A$
$I_{BZH}, I_{BZL}$	Bidirectional Pin Leakage Current	$GND \leq V_{IN} < V_{CC} \text{ MAX}$		—	$\pm 10$	$\mu A$
$I_{FZL}, I_{FZH}$	Output Pin Leakage Current	$GND \leq V_{IN} \leq V_{CC} \text{ MAX}$		—	$\pm 10$	$\mu A$
$I_{CC}$	Operating Power Supply Current	$F = 15 \text{ MHz}$ $V_{CC} = V_{CC} \text{ MAX}$	COM'L	—	45	mA
			MIL	—	50	mA
$I_{OS}^1$	Output Short Circuit	$V_{CC} = 5.0V, V_{OUT} = GND$		-30	-130	mA
$I_{SB}$	Standby Power Supply Current	$V_{CC} = V_{CC} \text{ MAX}$	COM'L	—	35	mA
			MIL	—	45	mA
$V_{OL}$	Output Low Voltage	$V_{CC} = V_{CC} \text{ MIN}$ $I_{OL} = 24 \text{ mA}$	COM'L	—	0.5	V
		$I_{OL} = 12 \text{ mA}$	MIL	—	0.5	V
$V_{OH}$	Output High Voltage	$V_{CC} = V_{CC} \text{ MIN}$ $I_{OH} = -3.2 \text{ mA}$	COM'L	2.4	—	V
		$I_{OH} = -2.0 \text{ mA}$	MIL	2.4	—	V
$V_{IH}$	Input High Voltage			2.0	$V_{CC} + 1$	V
$V_{IL}$	Input Low Voltage			—	0.8	V

**HALF-POWER GAL16V8**

SYMBOL	PARAMETER	TEST CONDITIONS	TEMP. RANGE	MIN.	MAX.	UNITS
$I_{IH}, I_{IL}$	Input Leakage Current	$GND \leq V_{IN} \leq V_{CC} \text{ MAX}$		—	$\pm 10$	$\mu A$
$I_{BZH}, I_{BZL}$	Bidirectional Pin Leakage Current	$GND \leq V_{IN} < V_{CC} \text{ MAX}$		—	$\pm 10$	$\mu A$
$I_{FZL}, I_{FZH}$	Output Pin Leakage Current	$GND \leq V_{IN} \leq V_{CC} \text{ MAX}$		—	$\pm 10$	$\mu A$
$I_{CC}$	Operating Power Supply Current	$F = 15 \text{ MHz}$ $V_{CC} = V_{CC} \text{ MAX}$	COM'L	—	90	mA
			MIL	—	90	mA
$I_{OS}^1$	Output Short Circuit	$V_{CC} = 5.0V, V_{OUT} = GND$		-30	-130	mA
$I_{SB}$	Standby Power Supply Current	$V_{CC} = V_{CC} \text{ MAX}$	COM'L	—	70	mA
			MIL	—	70	mA
$V_{OL}$	Output Low Voltage	$V_{CC} = V_{CC} \text{ MIN}$ $I_{OL} = 24 \text{ mA}$	COM'L	—	0.5	V
		$I_{OL} = 12 \text{ mA}$	MIL	—	0.5	V
$V_{OH}$	Output High Voltage	$V_{CC} = V_{CC} \text{ MIN}$ $I_{OH} = -3.2 \text{ mA}$	COM'L	2.4	—	V
		$I_{OH} = -2.0 \text{ mA}$	MIL	2.4	—	V
$V_{IH}$	Input High Voltage			2.0	$V_{CC} + 1$	V
$V_{IL}$	Input Low Voltage			—	0.8	V

<sup>1</sup> One output at a time for a maximum duration of one second.

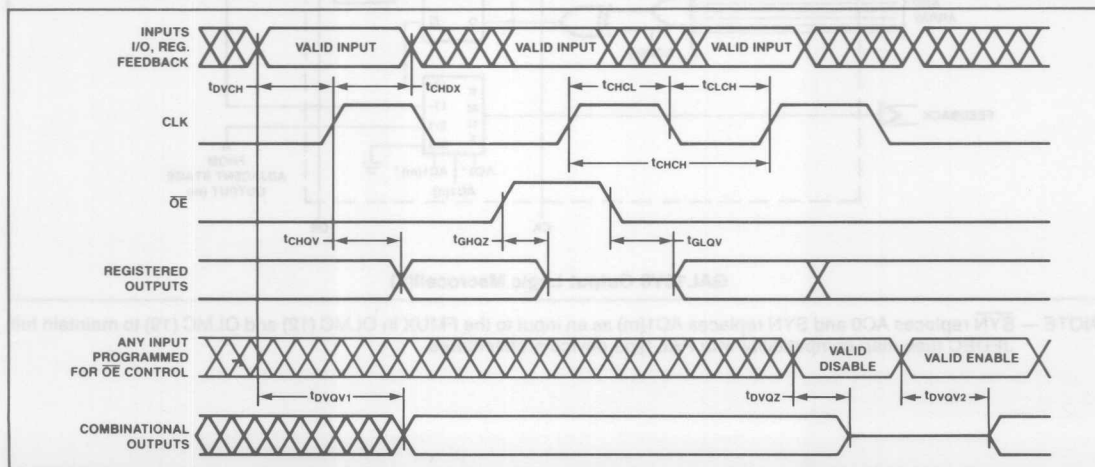


## SWITCHING CHARACTERISTICS OVER OPERATING CONDITIONS

		TEMPERATURE RANGE								
SYMBOL	PARAMETER	COMMERCIAL				MILITARY		UNITS	TEST CONDITIONS <sup>1</sup>	
		GAL16V8-25		GAL16V8-35		GAL16V8-30				
		MIN.	MAX.	MIN.	MAX.	MIN.	MAX.		R(Ω)	C <sub>L</sub> (pF)
T <sub>DVQV1</sub>	Delay from Input to Active Output	—	25	—	35	—	30	ns	200	50
T <sub>DVQV2</sub>	Product Term Enable Access Time to Active Output	—	25	—	35	—	30	ns	Active High R = ∞ Active Low R = 200	50
T <sub>DVQZ</sub> <sup>2</sup>	Product Term Disable to Outputs Off	—	25	—	35	—	30	ns	From V <sub>OH</sub> R = ∞ From V <sub>OL</sub> R = 200	5
T <sub>GHOZ</sub> <sup>2</sup>	OE Output Enable High to Outputs Off	—	20	—	25	—	25	ns	From V <sub>OH</sub> R = ∞ From V <sub>OL</sub> R = 200	5
T <sub>GLQV</sub>	OE Output Enable Access Time	—	20	—	25	—	25	ns	Active High R = ∞ Active Low R = 200	50
T <sub>CHQV</sub>	Clock High to Output Valid Access Time	—	15	—	25	—	20	ns	200	50
T <sub>DVCH</sub>	Input or Feedback Data Setup Time	20	—	30	—	25	—	ns	—	—
T <sub>CHDX</sub>	Input or Feedback Data Hold Time	0	—	0	—	0	—	ns	—	—
T <sub>CHCH</sub>	Clock Period (T <sub>DVCH</sub> + T <sub>CHQV</sub> )	35	—	55	—	45	—	ns		
T <sub>CHCL</sub>	Clock Width High	15	—	20	—	15	—	ns	—	—
T <sub>CLCH</sub>	Clock Width Low	15	—	20	—	15	—	ns		
f <sub>MAX</sub>	Maximum Frequency	SYNCH.	28.5		18.1	—	22.2	MHz	200	50
		ASYNCH.	40.0		28.5		33.3			

<sup>1</sup>Refer also to Switching Test Conditions. <sup>2</sup> 3-state levels are measured 0.5V from steady-state active level.

## SWITCHING WAVEFORMS



## OUTPUT LOGIC MACROCELL (OLMC)

The following discussion pertains to configuring the output logic macrocell. It should be noted that actual implementation is accomplished by development software/hardware and is completely transparent to the user.

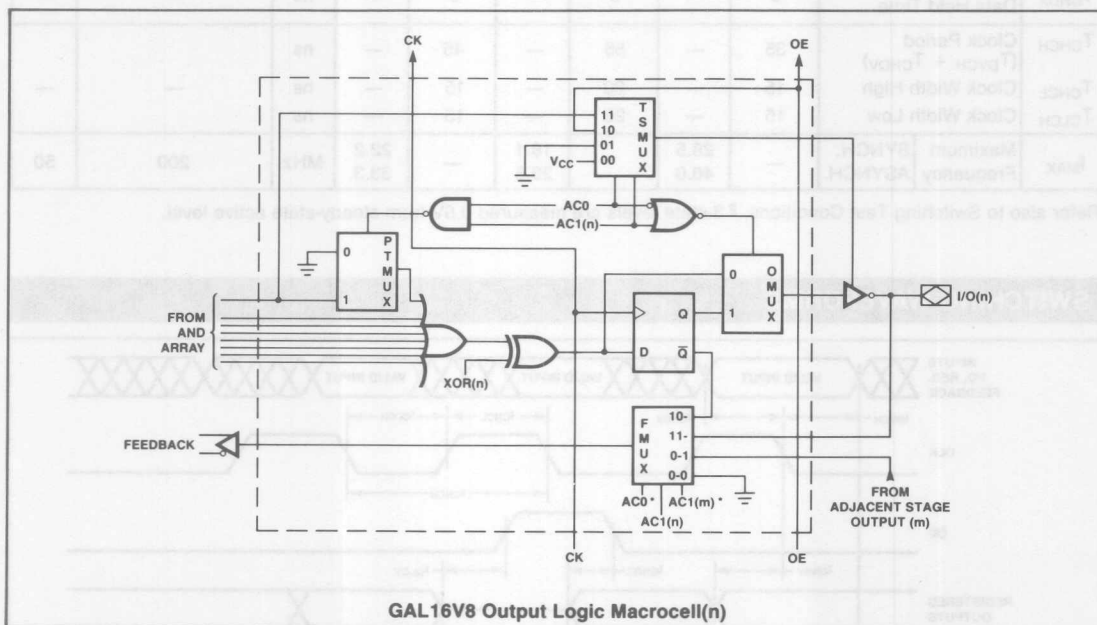
The outputs of the AND array are fed into an OLMC, where each output can be individually set to active high or active low, with either combinational (asynchronous) or registered (synchronous) configurations. A common output enable can be connected to all outputs, or separate inputs or product terms can be used to provide individual output enable controls. The output logic macrocell provides the designer with maximal output flexibility in matching signal requirements, thus providing more functions than possible with existing 20-pin PAL devices.

The various configurations of the output logic macrocell are controlled by programming certain cells (SYN, AC0, AC1(n), and the XOR(n) polarity bits) within the 82-bit architecture control word. The SYN bit determines

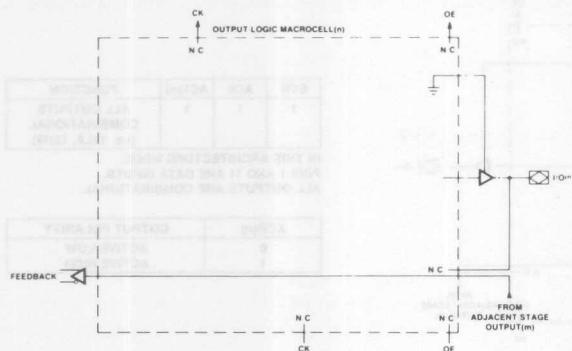
whether or not a device will have registered output capability or will have purely combinational outputs. It also replaces the AC0 bit in the two outermost macrocells, OLMC (12) and OLMC (19). When first setting up the device architecture, this is the first bit to choose.

Architecture control bit AC0 and the eight AC1(n) bits direct the outputs to be wired always on, always off (as an input), have a common  $\overline{OE}$  term (pin 11), or to be three-state controlled separately from a product term. The architecture control bits also determine the source of the array feedback term through the FMUX, and select either combinational or registered outputs.

The five valid macrocell configurations are shown in each of the macrocell equivalent diagrams. In all cases, the eight XOR(n) bits individually determine each output's polarity. The truth table associated with each diagram shows the bit values of the SYN, AC0, and AC1(n) that set the macrocell to the configuration shown.



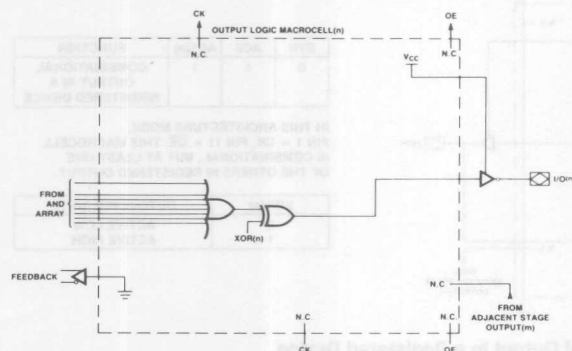
\* NOTE —  $\overline{SYN}$  replaces AC0 and SYN replaces AC1(m) as an input to the FMUX in OLMC (12) and OLMC (19) to maintain full JEDEC fuse map compatibility with PAL type device architectures.



SYN	AC0	AC1(n)	FUNCTION
1	0	1	INPUT MODE (i.e. 12L6, 14P4)

IN THIS ARCHITECTURE MODE,  
PINS 1 AND 11 ARE DATA INPUTS.  
THE OUTPUT BUFFER IS DISABLED.

Dedicated Input Mode

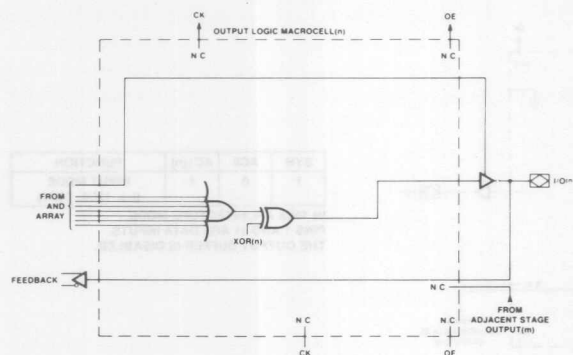


SYN	AC0	AC1(n)	FUNCTION
1	0	0	ALL OUTPUTS COMBINATIONAL (i.e. 10L8, 12H6)

IN THIS ARCHITECTURE MODE,  
PINS 1 AND 11 ARE DATA INPUTS.  
ALL OUTPUTS ARE COMBINATIONAL AND  
ALWAYS ACTIVE.

XOR(n)	OUTPUT POLARITY
0	ACTIVE LOW
1	ACTIVE HIGH

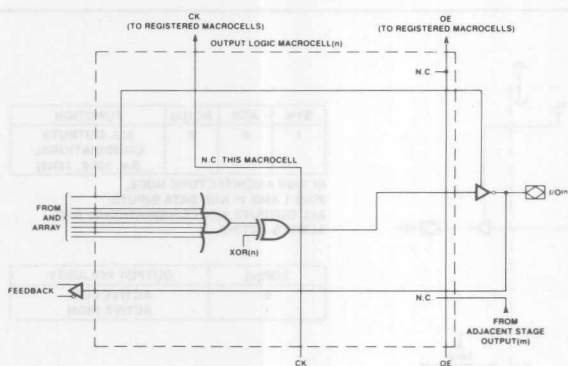
Dedicated Combinational Output


**Combinational Output**

SYN	AC0	AC1(n)	FUNCTION
1	1	1	ALL OUTPUTS COMBINATIONAL (i.e. 16L8, 16H8)

IN THIS ARCHITECTURE MODE,  
PINS 1 AND 11 ARE DATA INPUTS.  
ALL OUTPUTS ARE COMBINATIONAL.

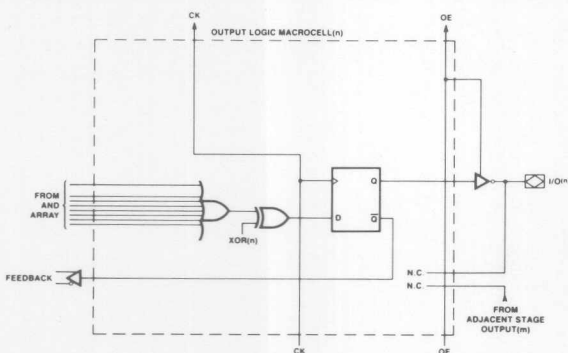
XOR(n)	OUTPUT POLARITY
0	ACTIVE LOW
1	ACTIVE HIGH


**Combinational Output in a Registered Device**

SYN	AC0	AC1(n)	FUNCTION
0	1	1	COMBINATIONAL OUTPUT IN A REGISTERED DEVICE

IN THIS ARCHITECTURE MODE,  
PIN 1 = CK, PIN 11 = OE. THIS MACROCELL  
IS COMBINATIONAL, BUT AT LEAST ONE  
OF THE OTHERS IS REGISTERED OUTPUT.

XOR(n)	OUTPUT POLARITY
0	ACTIVE LOW
1	ACTIVE HIGH


**Registered Active High or Low Output**

SYN	AC0	AC1(n)	FUNCTION
0	1	0	OUTPUT REGISTERED (i.e. 16R8)

IN THIS ARCHITECTURE MODE,  
PIN 1 = CK, PIN 11 = OE.

XOR(n)	OUTPUT POLARITY
0	ACTIVE LOW
1	ACTIVE HIGH



## ROW ADDRESS MAP DESCRIPTION

Figure 1 shows a block diagram of the row address map. There are a total of 36 unique row addresses available to the user when programming the GAL16V8 devices. Row addresses 0–31 each contain 64 bits of input term data. This is the user array where the custom logic pattern is programmed. Row 32 is the electronic signature word. It has 64 bits available for any user-defined purpose. Row 33–59 are reserved by the manufacturer and are not available to users.

Row 60 contains the architecture and output polarity information. The 82 bits within this word are programmed to configure the device for a specific application. Row 61 contains a one bit security cell that when programmed prevents further programming verification of the array. Row 63 is the row that is addressed to perform a bulk erase of the device, resetting it back to a virgin state. Each of these functions is described in the following sections.

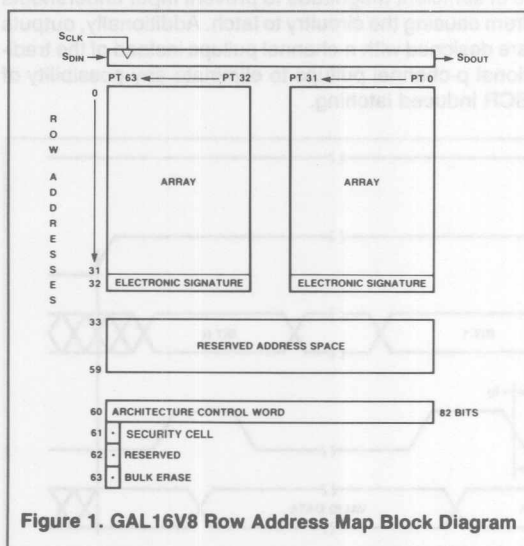


Figure 1. GAL16V8 Row Address Map Block Diagram

## ELECTRONIC SIGNATURE WORD

An electronic signature word is provided with every GAL16V8 device. It resides at Row address 32 and contains 64 bits of reprogrammable memory that can contain user-defined data. Some uses include user ID codes, revision numbers, or inventory control. This signature data is always available to the user independent of the state of the security cell.

## ARCHITECTURE CONTROL WORD

All of the various configurations of the GAL16V8 devices are controlled by programming cells within the 82-bit architecture control word that resides at row 60. The location of specific bits within the architecture control word is shown in the control word diagram in Figure 2. The function of the SYN, AC0 and AC1(n) bits have been explained in the output logic macrocell description. The eight polarity bits determine each output's polarity individually by selectively correct logic. The numbers below the XOR(n) and AC1(n) bits in the architecture control word diagram shows the output device pin number that the polarity bits control.

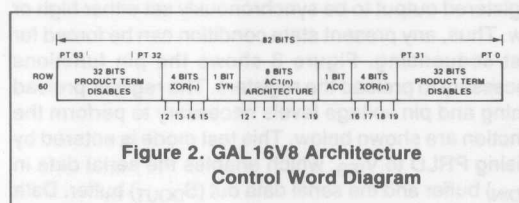


Figure 2. GAL16V8 Architecture Control Word Diagram

## SECURITY CELL

Row address 61 contains the security cell (one bit). The security cell is provided on all GAL16V8 devices as a deterrent to unauthorized copying of the array configuration patterns. Once programmed, the circuitry enabling array access is disabled, preventing further programming or verification of the array (rows 0–31). The cell can be erased only in conjunction with the array during a bulk erase cycle, so the original configuration can never be examined once this cell is programmed. Signature data is **always** available to the user.

## BULK ERASE MODE

By addressing row 63 during a programming cycle, a clear function performs a bulk erase of the array and the architecture word. In addition, the electronic signature word and the security cell are erased. This mode resets a previously configured device back to its virgin state.

## OUTPUT REGISTER PRELOAD

When testing state machine designs, all possible states and state transitions must be verified in the design, not just those required in the normal machine operations. This is because in system operation, certain events occur that may throw the logic into an illegal state (power-up, line voltage glitches, brown-outs, etc.). To test a design for proper treatment of these conditions, a way must be provided to break the feedback paths, and force any desired (ie. illegal) state into the registers. Then the machine can be sequenced and the outputs tested for correct next state conditions.

The GAL16V8 device includes circuitry that allows each registered output to be synchronously set either high or low. Thus, any present state condition can be forced for test sequencing. Figure 3 shows the pin functions necessary to preload the registers. The register preload timing and pin voltage levels necessary to perform the function are shown below. This test mode is entered by raising PRLD to  $V_{IES}$ , which enables the serial data in ( $S_{DIN}$ ) buffer and the serial data out ( $S_{DOUT}$ ) buffer. Data is then serially shifted into the registers on each rising edge of the clock, DCLK. Only the macrocells with registered output configurations are loaded. If only 3

outputs have registers, then only 3 bits need be shifted in. The registers are loaded from the bottom up, as shown in Figure 3.

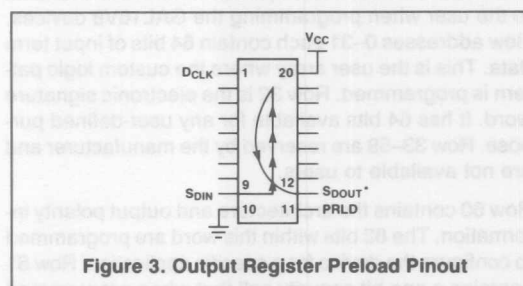


Figure 3. Output Register Preload Pinout

## LATCH-UP PROTECTION

GAL devices are designed with an on-board charge pump to negatively bias the substrate. The negative bias is of sufficient magnitude to prevent input undershoots from causing the circuitry to latch. Additionally, outputs are designed with n-channel pullups instead of the traditional p-channel pullups to eliminate any possibility of SCR induced latching.

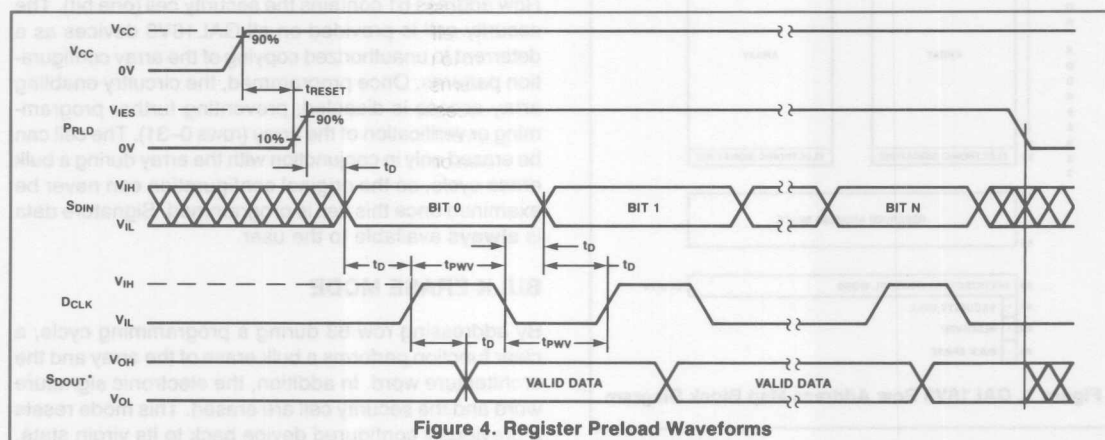
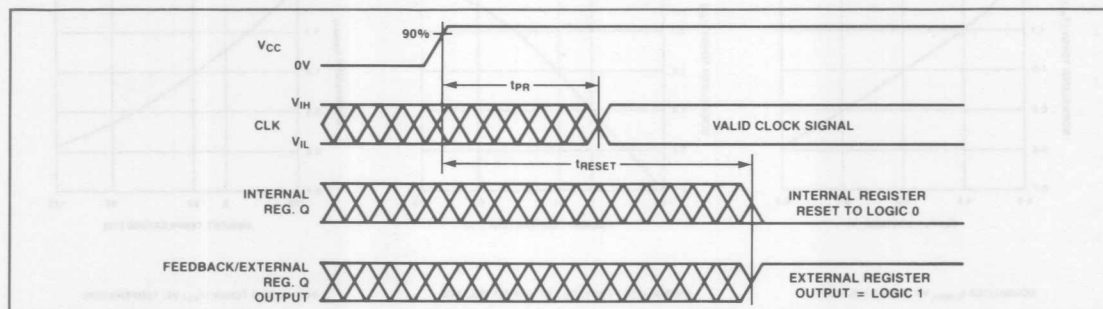


Figure 4. Register Preload Waveforms

\* NOTE — The  $S_{DOUT}$  output buffer is an open drain output during preload. This pin should be terminated to  $V_{CC}$  with a 10K resistor.

## POWER-UP RESET



Circuitry within the GAL16V8 provides a reset signal to all registers during power-up. All internal registers will have their Q outputs set low after a specified time ( $t_{RESET}$ ). As a result, the state on the registered output pins (if they are enabled through  $\overline{OE}$ ) will always be high on power-up, regardless of the programmed polarity of the output pins. This feature can greatly simplify state machine design by providing a known state on power-up.

The timing diagram for power-up is shown above. Because of asynchronous nature of system power-up, some conditions must be met to guarantee a valid power-up reset of the GAL16V8. First, the  $V_{CC}$  rise must be monotonic. Second, the clock input must become a proper TTL level within the specified time ( $t_{PR}$ ). The registers will reset within a maximum of  $t_{RESET}$  time. As in normal system operation, avoid clocking the device until all input and feedback path setup times have been met.

2

## FIELD SUPPORT TOOLS

## PROGRAMMER/DEVELOPMENT SYSTEMS

VENDOR	SYSTEM	REVISION
DATA I/O	Model 29B Logic Pak	V04
	P/T Adapter 303A-009	V03
	Model 60	**
STAG	PPZ/ZM2200	11/20
	ZL30 & ZL32	V30.41
	ZL30A	V30A.03
VARIX	Omni-Programmer	**
INLAB	Model 28	**
VALLEY DATA SCIENCES	Model 160	**

\*\* This version being qualified.

## SOFTWARE DEVELOPMENT TOOLST

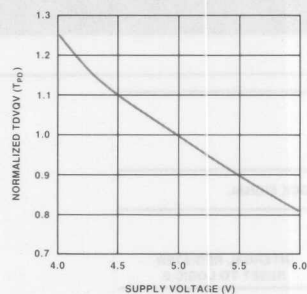
PACKAGE	VENDOR	REVISION
CUPL	Assisted Technology	2.1
ABEL	Data I/O	1.13
PLDtest	Data I/O	†
DASH-ABEL	Data I/O	1.0
PALASM	Monolithic Memories	†

† When emulating PAL devices any revision of the software can be used to create the PAL JEDEC file. The programming hardware will automatically configure the GAL architecture.

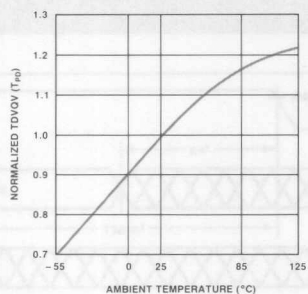
Although it is possible to program GAL devices manually, LATTICE strongly recommends the use of approved programming hardware and software. Programming on unapproved equipment will generally void all guarantees.

Approved equipment includes LATTICE programming algorithms that program the array, automatically configure the architecture control word, and track the number of program cycles each device has experienced (this information is stored within each GAL device). This in turn assures data retention and reliability. Contact the factory for specific conditions which must be met to gain programming equipment approval or for the current list of approved GAL programming equipment.

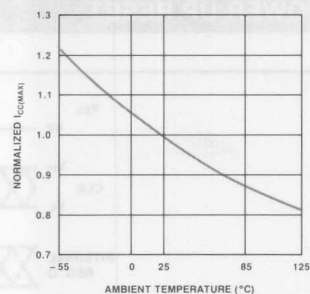
NORMALIZED TDQV ( $T_{PD}$ ) VS. SUPPLY VOLTAGE



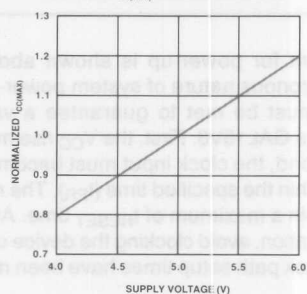
NORMALIZED TDQV ( $T_{PD}$ ) VS. TEMPERATURE



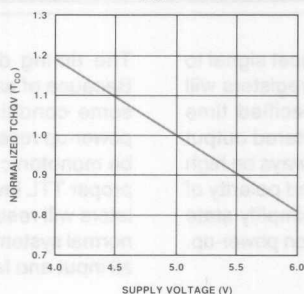
NORMALIZED  $I_{CC(MAX)}$  VS. TEMPERATURE



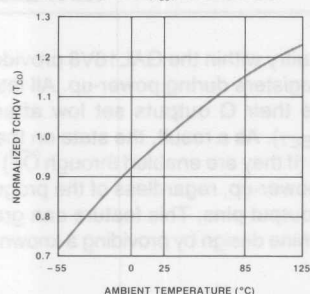
NORMALIZED  $I_{CC(MAX)}$  VS. SUPPLY VOLTAGE



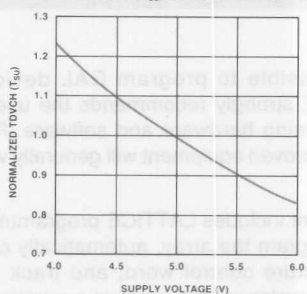
NORMALIZED TCHOV ( $T_{CO}$ ) VS. SUPPLY VOLTAGE



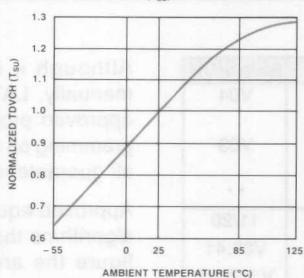
NORMALIZED TCHOV ( $T_{CO}$ ) VS. TEMPERATURE



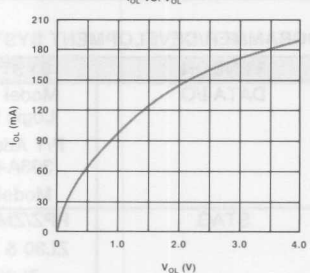
NORMALIZED TDVCH ( $T_{SU}$ ) VS. SUPPLY VOLTAGE



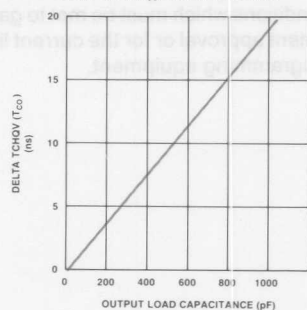
NORMALIZED TDVCH ( $T_{SU}$ ) VS. TEMPERATURE



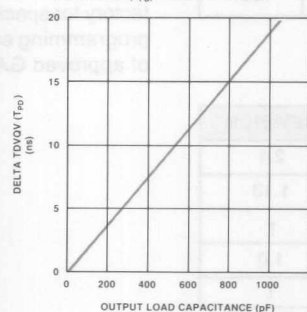
$I_{OL}$  VS.  $V_{OL}$



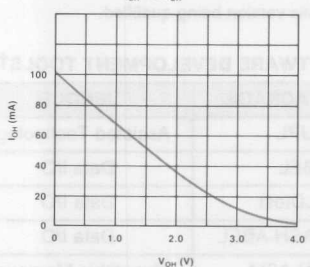
DELTA TCHOV ( $T_{CO}$ ) VS. OUTPUT LOADING



DELTA TDQV ( $T_{PD}$ ) VS. OUTPUT LOADING



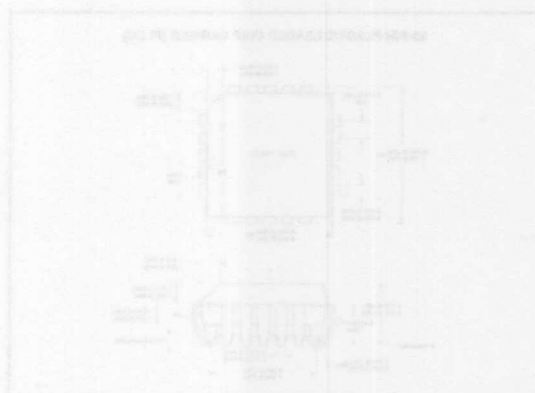
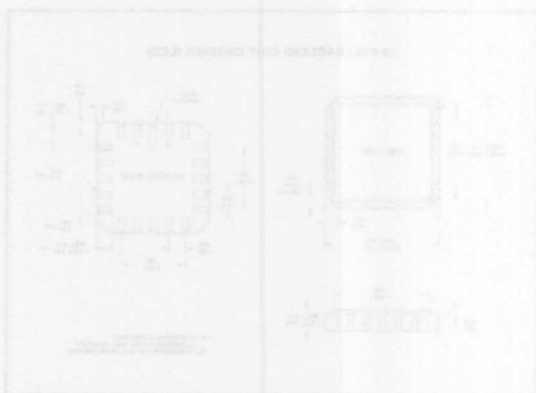
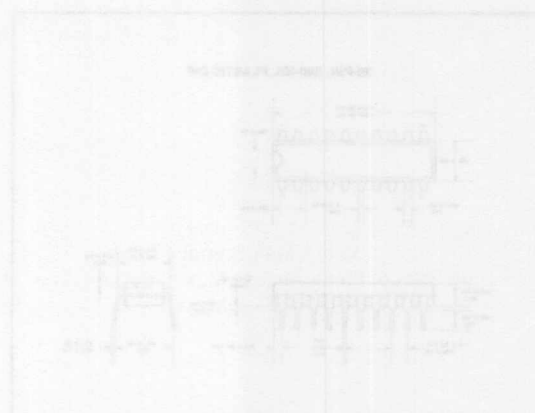
$I_{OH}$  VS.  $V_{OH}$







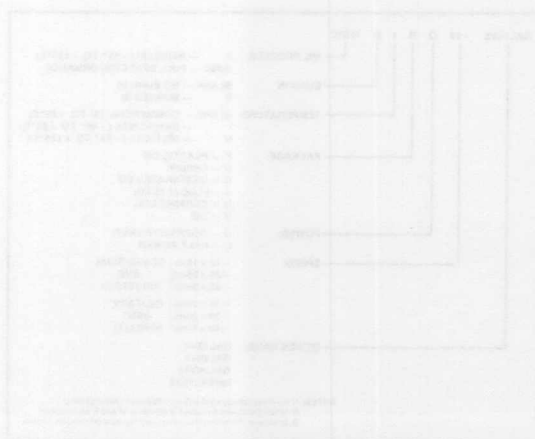
PACKAGE INFORMATION



ORDERING INFORMATION

TEMPERATURE RANGE	LOGIC FUNCTION	PACKAGE	QUANTITY
0°C to 70°C	001 - 001	001	001
0°C to 70°C	002 - 002	002	002
0°C to 70°C	003 - 003	003	003
0°C to 70°C	004 - 004	004	004
0°C to 70°C	005 - 005	005	005
0°C to 70°C	006 - 006	006	006
0°C to 70°C	007 - 007	007	007
0°C to 70°C	008 - 008	008	008
0°C to 70°C	009 - 009	009	009
0°C to 70°C	010 - 010	010	010
0°C to 70°C	011 - 011	011	011
0°C to 70°C	012 - 012	012	012
0°C to 70°C	013 - 013	013	013
0°C to 70°C	014 - 014	014	014
0°C to 70°C	015 - 015	015	015
0°C to 70°C	016 - 016	016	016
0°C to 70°C	017 - 017	017	017
0°C to 70°C	018 - 018	018	018
0°C to 70°C	019 - 019	019	019
0°C to 70°C	020 - 020	020	020

ORDERING INFORMATION



## FEATURES

- ELECTRICALLY ERASABLE CELL TECHNOLOGY
  - Reconfigurable Logic
  - Reprogrammable Cells
  - Guaranteed 100% Yields
- HIGH PERFORMANCE E<sup>2</sup>CMOS TECHNOLOGY
  - Low Power: 45 mA Max Active  
35 mA Max Standby
  - High Speed: 25 ns Access Max  
35 ns Access Max
- EIGHT OUTPUT LOGIC MACROCELLS
  - Maximum Flexibility for Complex Logic Designs
  - Also Emulates 24-Pin PAL<sup>®</sup> Devices with Full Function/Fuse Map Compatibility
- PRELOAD AND POWER-ON RESET OF ALL REGISTERS
  - 100% Functional Testability
- HIGH SPEED PROGRAMMING ALGORITHM
- SECURITY CELL PREVENTS COPYING LOGIC
- DATA RETENTION EXCEEDS 20 YEARS

## DESCRIPTION

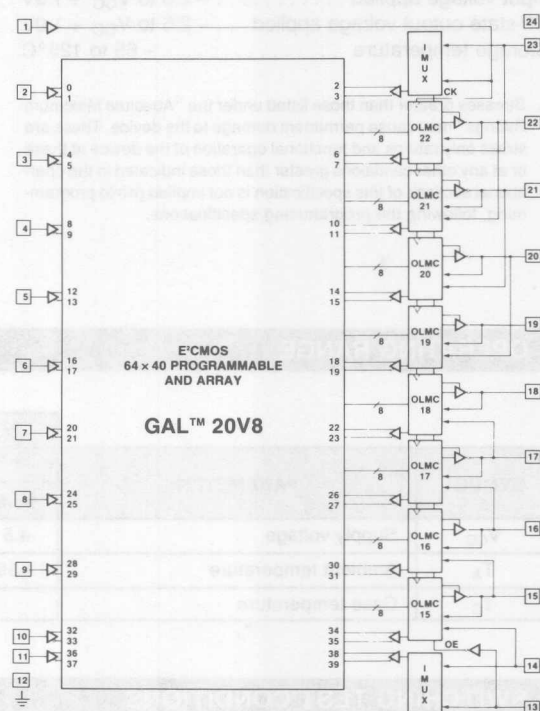
Lattice Semiconductor's E<sup>2</sup>CMOS GAL20V8 combines a high performance CMOS process with electrically erasable floating gate technology. This programmable memory technology applied to array logic provides designers with reconfigurable logic and bipolar performance at significantly reduced power levels.

The 24-pin GAL20V8 features 8 programmable Output Logic Macrocells (OLMC) allowing each output to be configured by the user. Additionally, the GAL20V8 is capable of emulating, in a functional/fuse map compatible device, all common 24-pin PAL device architectures.

Programming is accomplished using readily available hardware and software tools. LATTICE guarantees 100 erase/write cycles, and that data retention exceeds 20 years.

Unique test circuitry and programmable cells allow complete AC, DC, cell and functionality testing during manufacture. Therefore, LATTICE guarantees 100% field programmability and functionality of all GAL products. In addition, electronic signature is available to provide positive device ID. A security circuit is built-in, providing proprietary designs with copy protection.

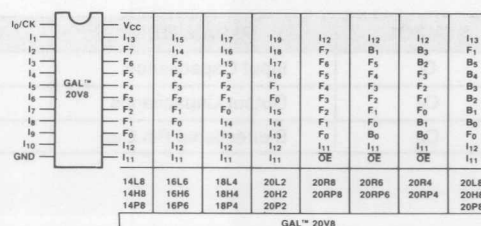
## FUNCTIONAL BLOCK DIAGRAM



## PIN NAMES

I <sub>0</sub> -I <sub>19</sub>	INPUT	$\overline{OE}$	OUTPUT ENABLE
CK	CLOCK INPUT	V <sub>CC</sub>	POWER (+5V)
B <sub>0</sub> -B <sub>5</sub>	BIDIRECTIONAL	GND	GROUND
F <sub>0</sub> -F <sub>7</sub>	OUTPUT		

## GAL20V8 EMULATING PAL DEVICES

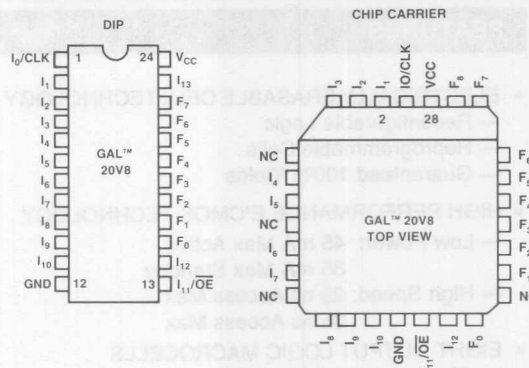


## ABSOLUTE MAXIMUM RATINGS 1

Supply voltage  $V_{CC}$  ..... - .5 to +7V  
Input voltage applied ..... -2.5 to  $V_{CC}$  +1.0V  
Off-state output voltage applied ... -2.5 to  $V_{CC}$  +1.0V  
Storage temperature ..... -65 to 125°C

- Stresses greater than those listed under the "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress only ratings and functional operation of the device at these or at any other conditions greater than those indicated in the operational sections of this specification is not implied (while programming, following the programming specifications).

## PIN CONFIGURATION



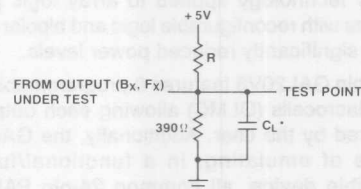
## OPERATING RANGE

		TEMPERATURE RANGE						
SYMBOL	PARAMETER	MILITARY			COMMERCIAL			UNIT
		MIN	TYP	MAX	MIN	TYP	MAX	
V <sub>CC</sub>	Supply voltage	4.5	5	5.5	4.75	5	5.25	V
T <sub>A</sub>	Ambient temperature	− 55			0		75	°C
T <sub>C</sub>	Case temperature			125				°C

## SWITCHING TEST CONDITIONS

Input Pulse Levels	GND to 3.0V
Input Rise and Fall Times	5ns 10% - 90%
Input Timing Reference Levels	1.5V
Output Timing Reference Levels	1.5V
Output Load	See Figure

3-state levels are measured 0.5V from steady-state active level.



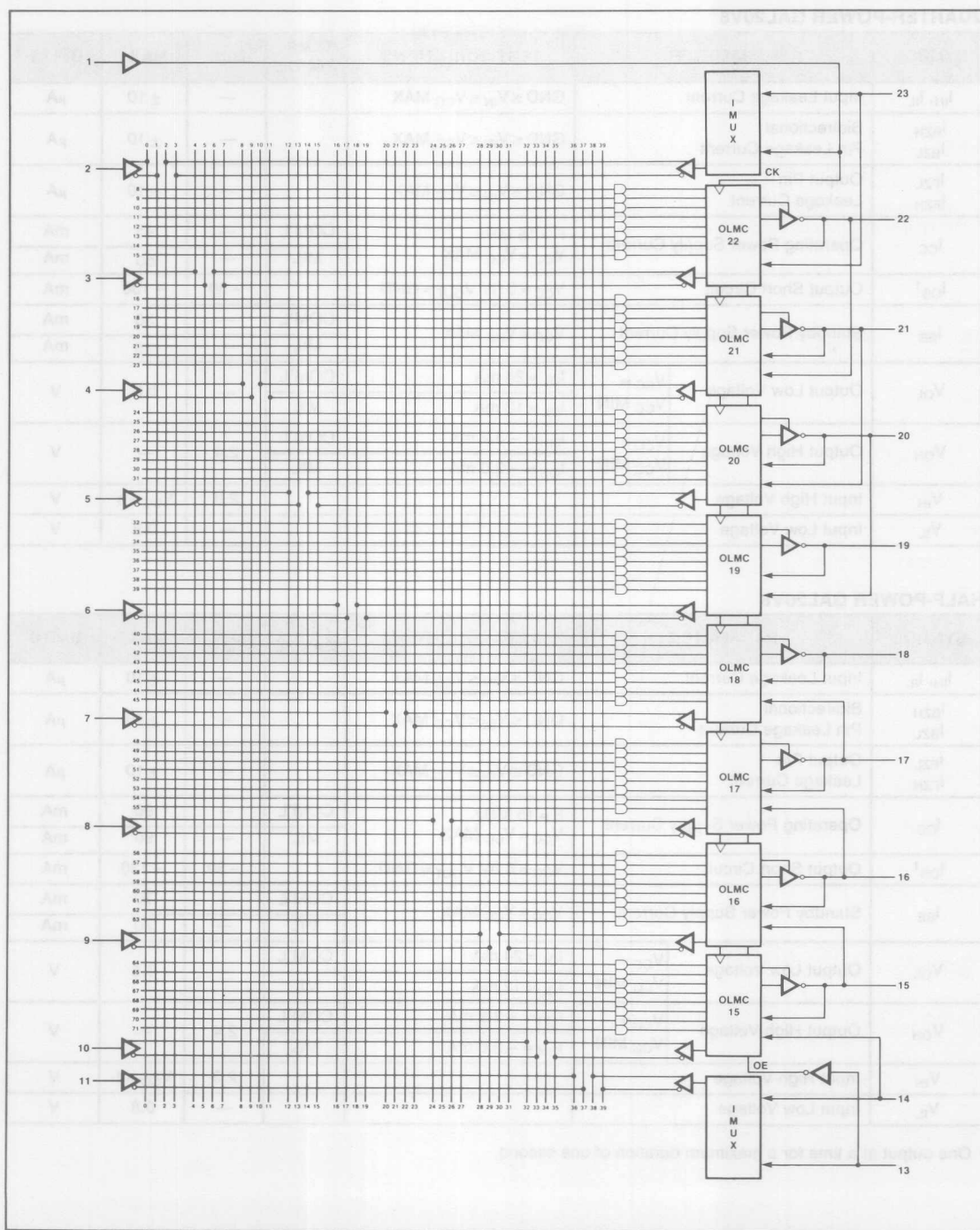
\* $C_L$  INCLUDES JIG AND PROBE TOTAL CAPACITANCE

## CAPACITANCE ( $T_A = 25^\circ\text{C}$ , $f = 1.0\text{ MHz}$ )

SYMBOL	PARAMETER	MAXIMUM	UNITS	TEST CONDITIONS
$C_I$	Input Capacitance	12	pF	$V_{CC} = 5.0\text{V}$ , $V_I = 2.0\text{V}$
$C_F$	Output Capacitance	15	pF	$V_{CC} = 5.0\text{V}$ , $V_F = 2.0\text{V}$
$C_B$	Bidirectional Pin Cap	15	pF	$V_{CC} = 5.0\text{V}$ , $V_B = 2.0\text{V}$



GAL 20V8 LOGIC DIAGRAM



**ELECTRICAL CHARACTERISTICS OVER OPERATING CONDITIONS**
**PRELIMINARY**
**QUARTER-POWER GAL20V8**

SYMBOL	PARAMETER	TEST CONDITIONS	TEMP. RANGE	MIN.	MAX.	UNITS
$I_{IH}, I_{IL}$	Input Leakage Current	$GND \leq V_{IN} \leq V_{CC} \text{ MAX}$		—	$\pm 10$	$\mu A$
$I_{BZH}, I_{BZL}$	Bidirectional Pin Leakage Current	$GND \leq V_{IN} < V_{CC} \text{ MAX}$		—	$\pm 10$	$\mu A$
$I_{FZL}, I_{FZH}$	Output Pin Leakage Current	$GND \leq V_{IN} \leq V_{CC} \text{ MAX}$		—	$\pm 10$	$\mu A$
$I_{CC}$	Operating Power Supply Current	$F = 15 \text{ MHz}$ $V_{CC} = V_{CC} \text{ MAX}$	COM'L	—	45	mA
			MIL	—	50	mA
$I_{OS}^1$	Output Short Circuit	$V_{CC} = 5.0V, V_{OUT} = GND$		-30	-130	mA
$I_{SB}$	Standby Power Supply Current	$V_{CC} = V_{CC} \text{ MAX}$	COM'L	—	35	mA
			MIL	—	45	mA
$V_{OL}$	Output Low Voltage	$V_{CC} = V_{CC} \text{ MIN}$ $I_{OL} = 24 \text{ mA}$ $I_{OL} = 12 \text{ mA}$	COM'L	—	0.5	V
			MIL			
$V_{OH}$	Output High Voltage	$V_{CC} = V_{CC} \text{ MIN}$ $I_{OH} = -3.2 \text{ mA}$ $I_{OH} = -2.0 \text{ mA}$	COM'L	2.4	—	V
			MIL			
$V_{IH}$	Input High Voltage			2.0	$V_{CC} + 1$	V
$V_{IL}$	Input Low Voltage			—	0.8	V

**HALF-POWER GAL20V8**

SYMBOL	PARAMETER	TEST CONDITIONS	TEMP. RANGE	MIN.	MAX.	UNITS
$I_{IH}, I_{IL}$	Input Leakage Current	$GND \leq V_{IN} \leq V_{CC} \text{ MAX}$		—	$\pm 10$	$\mu A$
$I_{BZH}, I_{BZL}$	Bidirectional Pin Leakage Current	$GND \leq V_{IN} < V_{CC} \text{ MAX}$		—	$\pm 10$	$\mu A$
$I_{FZL}, I_{FZH}$	Output Pin Leakage Current	$GND \leq V_{IN} \leq V_{CC} \text{ MAX}$		—	$\pm 10$	$\mu A$
$I_{CC}$	Operating Power Supply Current	$F = 15 \text{ MHz}$ $V_{CC} = V_{CC} \text{ MAX}$	COM'L	—	90	mA
			MIL	—	90	mA
$I_{OS}^1$	Output Short Circuit	$V_{CC} = 5.0V, V_{OUT} = GND$		-30	-130	mA
$I_{SB}$	Standby Power Supply Current	$V_{CC} = V_{CC} \text{ MAX}$	COM'L	—	70	mA
			MIL	—	70	mA
$V_{OL}$	Output Low Voltage	$V_{CC} = V_{CC} \text{ MIN}$ $I_{OL} = 24 \text{ mA}$ $I_{OL} = 12 \text{ mA}$	COM'L	—	0.5	V
			MIL			
$V_{OH}$	Output High Voltage	$V_{CC} = V_{CC} \text{ MIN}$ $I_{OH} = -3.2 \text{ mA}$ $I_{OH} = -2.0 \text{ mA}$	COM'L	2.4	—	V
			MIL			
$V_{IH}$	Input High Voltage			2.0	$V_{CC} + 1$	V
$V_{IL}$	Input Low Voltage			—	0.8	V

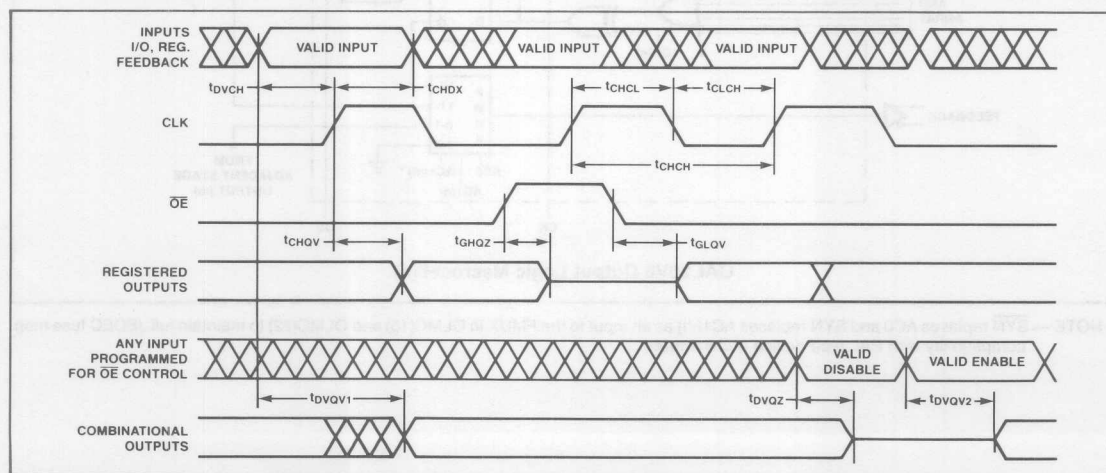
<sup>1</sup> One output at a time for a maximum duration of one second.

## SWITCHING CHARACTERISTICS OVER OPERATING CONDITIONS

SYMBOL			PARAMETER		TEMPERATURE RANGE						UNITS	TEST CONDITIONS <sup>1</sup>	
					COMMERCIAL				MILITARY				
					GAL20V8-25		GAL20V8-35		GAL20V8-30				
					MIN.	MAX.	MIN.	MAX.	MIN.	MAX.		R(Ω)	C <sub>L</sub> (pF)
T <sub>DVQV1</sub>			Delay from Input to Active Output		—	25	—	35	—	30	ns	200	50
T <sub>DVQV2</sub>			Product Term Enable Access Time to Active Output		—	25	—	35	—	30	ns	Active High R = ∞ Active Low R = 200	50
T <sub>DVQZ</sub> <sup>2</sup>			Product Term Disable to Outputs Off		—	25	—	35	—	30	ns	From V <sub>OH</sub> R = ∞ From V <sub>OL</sub> R = 200	5
T <sub>GHQZ</sub> <sup>2</sup>			OE (Output Enable) High to Outputs Off		—	20	—	25	—	25	ns	From V <sub>OH</sub> R = ∞ From V <sub>OL</sub> R = 200	5
T <sub>GLQV</sub>			OE (Output Enable) Access Time		—	20	—	25	—	25	ns	Active High R = ∞ Active Low R = 200	50
T <sub>CHQV</sub>			Clock High to Output Valid Access Time		—	15	—	25	—	20	ns	200	50
T <sub>DVCH</sub>			Input or Feedback Data Setup Time		20	—	30	—	25	—	ns	—	—
T <sub>CHDX</sub>			Input or Feedback Data Hold Time		0	—	0	—	0	—	ns	—	—
T <sub>CHCH</sub>			Clock Period (T <sub>DVCH</sub> + T <sub>CHQV</sub> )		35	—	55	—	45	—	ns	—	—
T <sub>CHCL</sub>			Clock Width High		15	—	20	—	15	—	ns		
T <sub>CLCH</sub>			Clock Width Low		15	—	20	—	15	—	ns		
f <sub>MAX</sub>		Maximum Frequency	SYNCH. ASYNCH.	—	28.5 40.0	—	18.1 28.5	—	22.2 33.3	MHz	200	50	

<sup>1</sup>Refer also to Switching Test Conditions. <sup>2</sup>3-state levels are measured 0.5V from steady-state active level

## SWITCHING WAVEFORMS



## OUTPUT LOGIC MACROCELL

The following discussion pertains to configuring the Output Logic Macrocell. It should be noted that actual implementation is accomplished by development software/hardware and is completely transparent to the user.

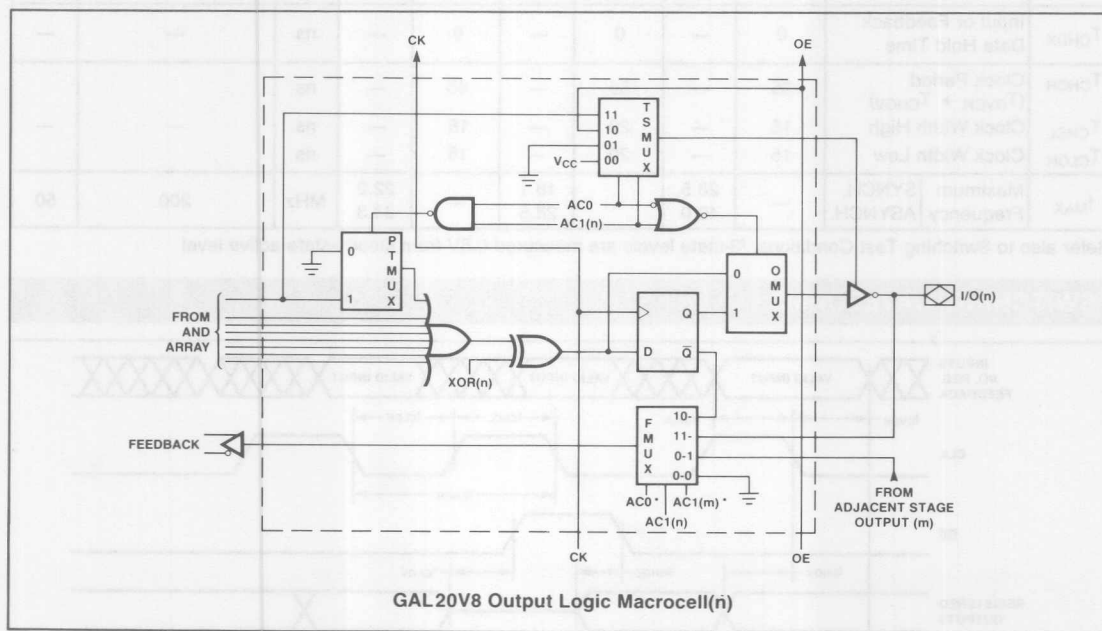
The outputs of the AND array are fed into an OLMC, where each output can be individually set to active high, or active low, with either combinational (asynchronous) or registered (synchronous) configurations. A common output enable can be connected to all outputs, or separate inputs or product terms can be used to provide individual output enable controls. The Output Logic Macrocell provides the designer with maximal output flexibility in matching signal requirements, thus providing more functions than possible with existing 24-pin PAL devices.

The various configurations of the Output Logic Macrocell are controlled by programming certain cells (SYN, AC0, AC1(n), and the XOR(n) polarity bits) within the 82-bit Architecture Control Word. The SYN bit determines

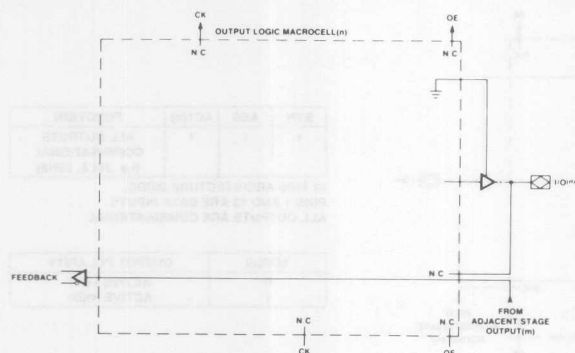
whether a device will have registered output capability or purely combinational outputs. It also replaces the AC0 bit in the two outermost macrocells, OLMC (15) and OLMC (22). When first setting up the device architecture, this is the first bit to choose.

Architecture Control bit AC0 and the eight AC1(n) bits direct the outputs to be wired always on, always off (as an input), have a common OE term (pin 13), or to be three-state controlled separately from a product term. The Architecture Control bits also determine the source of the array feedback term through the FMUX, and select either combinational or registered outputs.

The five valid macrocell configurations are shown in each of the macrocell equivalent diagrams. In all cases, the eight XOR(n) bits individually determine each output's polarity. The truth table associated with each diagram shows the bit values of the SYN, AC0, and AC1(n) that set the macrocell to the configuration shown.



\* NOTE —  $\overline{\text{SYN}}$  replaces AC0 and SYN replaces AC1(m) as an input to the FMUX in OLMC(15) and OLMC(22) to maintain full JEDEC fuse map compatibility with PAL type device architectures.

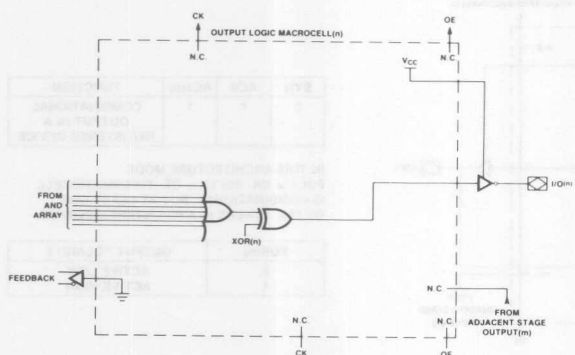


SYN	AC0	AC1(n)	FUNCTION
1	0	1	INPUT MODE (i.e. 20L2, 18P4)

IN THIS ARCHITECTURE MODE,  
PINS 1 AND 13 ARE DATA INPUTS.  
THE OUTPUT BUFFER IS DISABLED.

Dedicated Input Mode

2



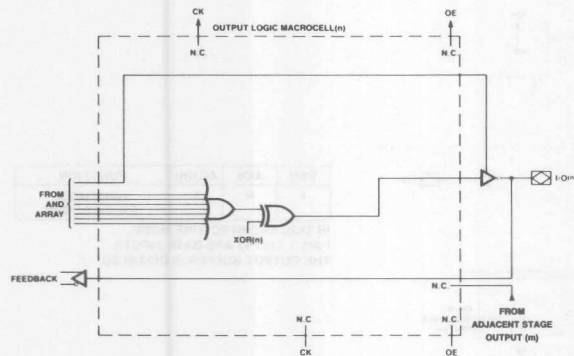
SYN	AC0	AC1(n)	FUNCTION
1	0	0	ALL OUTPUTS COMBINATIONAL (i.e. 16H6)

IN THIS ARCHITECTURE MODE,  
PINS 1 AND 13 ARE DATA INPUTS.  
ALL MACROCELLS CONFIGURED AS OUTPUTS  
ARE COMBINATIONAL AND ALWAYS ACTIVE.

XOR(n)	OUTPUT POLARITY
0	ACTIVE LOW
1	ACTIVE HIGH

Dedicated Combinational Output

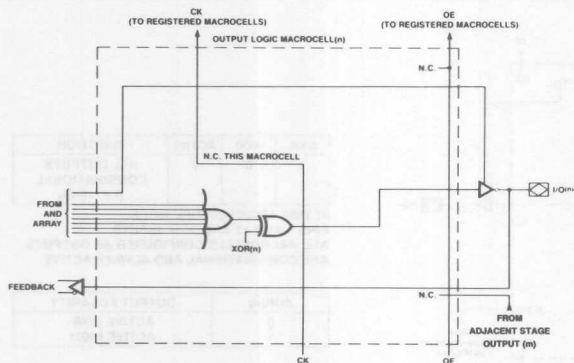



**Combinational Output**

SYN	AC0	AC1(n)	FUNCTION
1	1	1	ALL OUTPUTS COMBINATIONAL (i.e. 20L8, 20H8)

IN THIS ARCHITECTURE MODE, PINS 1 AND 13 ARE DATA INPUTS. ALL OUTPUTS ARE COMBINATIONAL.

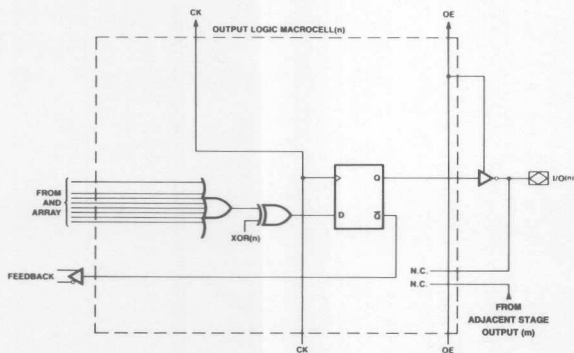
XOR(n)	OUTPUT POLARITY
0	ACTIVE LOW
1	ACTIVE HIGH


**Combinational Output in a Registered Device**

SYN	AC0	AC1(n)	FUNCTION
0	1	1	COMBINATIONAL OUTPUT IN A REGISTERED DEVICE

IN THIS ARCHITECTURE MODE, PIN 1 = CK, PIN 13 = OE. THIS MACROCELL IS COMBINATIONAL, BUT AT LEAST ONE OF THE OTHERS IS A REGISTERED OUTPUT.

XOR(n)	OUTPUT POLARITY
0	ACTIVE LOW
1	ACTIVE HIGH


**Registered Active High or Low Output**

SYN	AC0	AC1(n)	FUNCTION
0	1	0	OUTPUT REGISTERED (i.e. 20R8)

IN THIS ARCHITECTURE MODE, PIN 1 = CK, PIN 13 = OE.

XOR(n)	OUTPUT POLARITY
0	ACTIVE LOW
1	ACTIVE HIGH

## ROW ADDRESS MAP DESCRIPTION

Figure 1 shows a block diagram of the row address map. There are a total of 44 unique row addresses available to the user when programming the GAL20V8. Row addresses 0-39 each contain 64 bits of input term data. This is the user array where the custom logic pattern is programmed. Row 40 is the Electronic Signature Word. It has 64 bits available for any user-defined purpose. Row 41-59 are reserved by the manufacturer and are not available to users.

Row 60 contains the architecture and output polarity information. The 82 bits within this word are programmed to configure the device for a specific application. Row 61 contains a one bit Security Cell that when programmed prevents further programming verification of the array. Row 63 is the row that is addressed to perform a bulk erase of the device, resetting it to a virgin state. Each of these functions is described in the following sections.

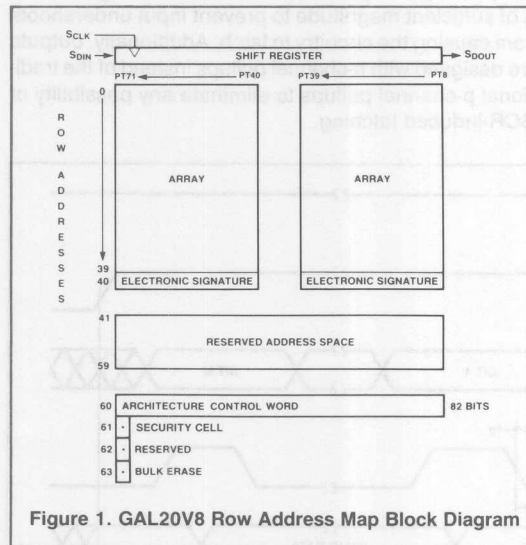


Figure 1. GAL20V8 Row Address Map Block Diagram

## ELECTRONIC SIGNATURE WORD

An Electronic Signature Word is provided with every GAL20V8 device. It resides at row address 40 and contains 64 bits of reprogrammable memory that can contain user-defined data. Some uses include ID codes, revision numbers, or inventory control. The ability to mark and identify parts electronically means improved parts handling and lower inventory costs. This signature data is always available to the user independent of the state of the Security Cell.

## ARCHITECTURE CONTROL WORD

All of the various configurations of the GAL20V8 are controlled by programming cells within the 82-bit Architecture Control Word that resides at row 60. The location of specific bits within the Architecture Control Word is shown in the control word diagram in Figure 2. The function of the SYN, AC0 and AC1(n) bits have been explained in the Output Logic Macrocell description. The eight polarity bits determine each output's polarity individually. The numbers below the XOR(n) and AC1(n) bits in Figure 2 show the output device pin numbers that the polarity bits control.

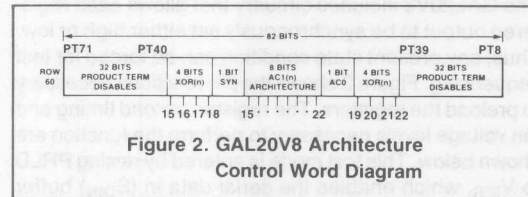


Figure 2. GAL20V8 Architecture Control Word Diagram

## SECURITY CELL

Row address 61 contains the Security Cell (one bit). The Security Cell is provided on all GAL20V8 devices as a deterrent to unauthorized copying of the array configuration patterns. Once programmed, the circuitry enabling array access is disabled, preventing further programming or verification of the AND array (rows 0-39). The cell can be erased only in conjunction with the array during a bulk erase cycle, so the original configuration can never be examined once this cell is programmed. Signature data is **always** available to the user.

## BULK ERASE MODE

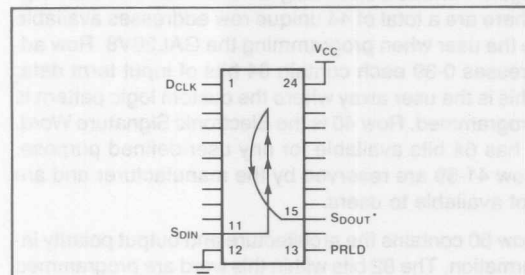
By addressing row 63 during a programming cycle, a clear function performs a bulk erase of the array and the Architecture Control Word. In addition, the Electronic Signature Word and the Security Cell are erased. This mode resets a previously configured device to its virgin state.

## OUTPUT REGISTER PRELOAD

When testing state machine designs, all possible states and state transitions must be verified in the design, not just those required in the normal machine operations. This is because in system operation, certain events occur that may throw the logic into an illegal state (power-up, line voltage glitches, brown-outs, etc.). To test a design for proper treatment of these conditions, a way must be provided to break the feedback paths, and force any desired (ie. illegal) state into the registers. Then the machine can be sequenced and the outputs tested for correct next state conditions.

The GAL20V8 includes circuitry that allows each registered output to be synchronously set either high or low. Thus, any present state condition can be forced for test sequencing. Figure 3 shows the pin functions necessary to preload the registers. The register preload timing and pin voltage levels necessary to perform the function are shown below. This test mode is entered by raising PRLD to  $V_{IES}$ , which enables the serial data in ( $S_{DIN}$ ) buffer and the serial data out ( $S_{DOUT}$ ) buffer. Data is then serially shifted into the registers on each rising edge of the clock, DCLK. Only the macrocells with registered output configurations are loaded. If only 3 outputs have reg-

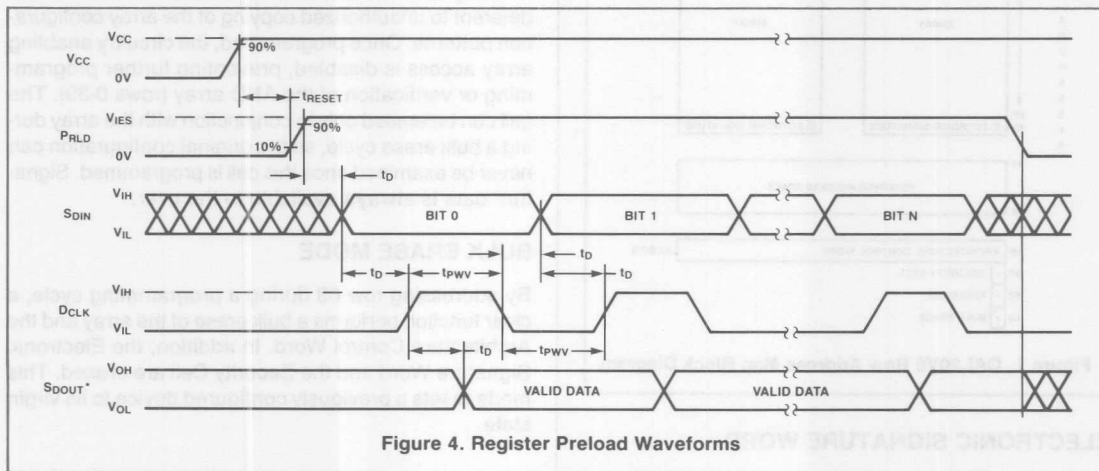
isters, then only 3 bits, need be shifted in. The registers are loaded from the bottom up, as shown in Figure 3.



**Figure 3. Output Register Preload Pinout**

## LATCH-UP PROTECTION

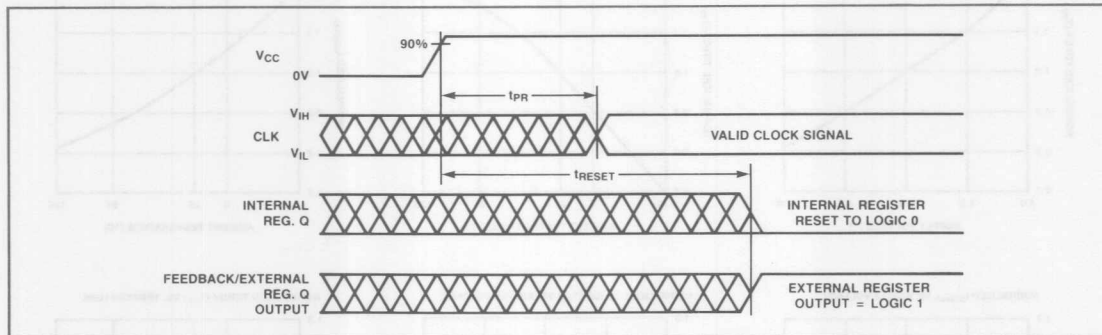
GAL devices are designed with an on-board charge pump to negatively bias the substrate. The negative bias is of sufficient magnitude to prevent input undershoots from causing the circuitry to latch. Additionally, outputs are designed with n-channel pullups instead of the traditional p-channel pullups to eliminate any possibility of SCR-induced latching.



**Figure 4. Register Preload Waveforms**

\*NOTE — The  $S_{DOUT}$  output buffer is an open drain output during preload. This pin should be terminated to  $V_{CC}$  with a 10K resistor.

## POWER-UP RESET



Circuitry within the GAL20V8 provides a reset signal to all registers during power-up. All internal registers will have their Q outputs set low after a specified time ( $t_{RESET}$ ). As a result, the state on the registered output pins (if they are enabled through OE) will always be high on power-up, regardless of the programmed polarity of the output pins. This feature can greatly simplify state machine design by providing a known state on power-up.

The timing diagram for power-up is shown above. Because of the asynchronous nature of system power-up, some conditions must be met to guarantee a valid power-up reset of the GAL20V8. First, the  $V_{CC}$  rise must be monotonic. Second, the clock input must become a proper TTL level within the specified time ( $t_{PR}$ ). The registers will reset within a maximum of  $t_{RESET}$  time. As in normal system operation, avoid clocking the device until all input and feedback path setup times have been met.

2

## FIELD SUPPORT TOOLS

VENDOR	SYSTEM	REVISION
DATA I/O	Model 29B Logic Pak	V04
	P/T Adapter 303A-009	V03
	Model 60	**
STAG	PPZ/ZM2200	11/20
	ZL30 & ZL32	V30.41
	ZL30A	V30A.03
VARIX	Omni-Programmer	**
INLAB	Model 28	**
VALLEY DATA SCIENCES	Model 160	**

\*\* This version being qualified.

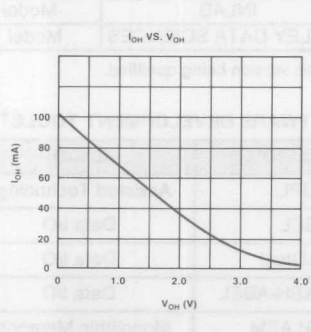
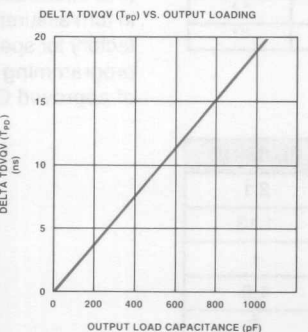
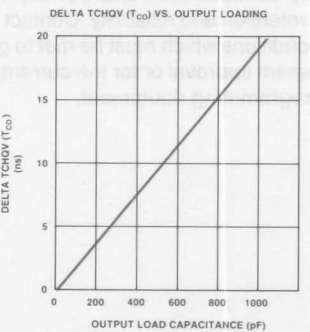
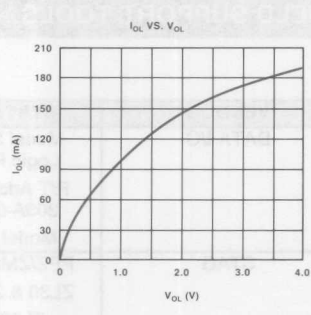
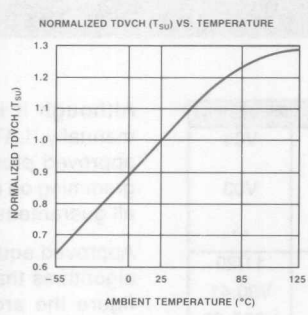
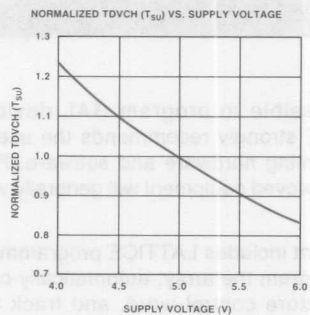
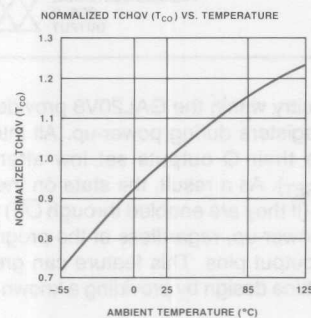
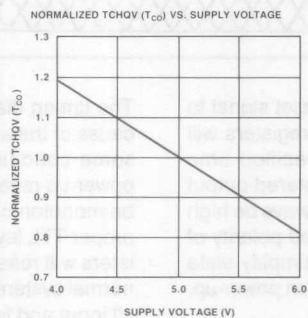
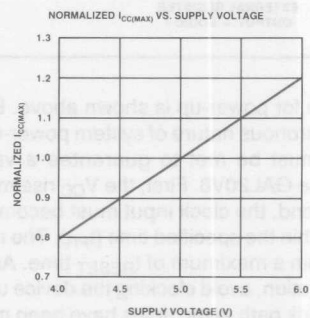
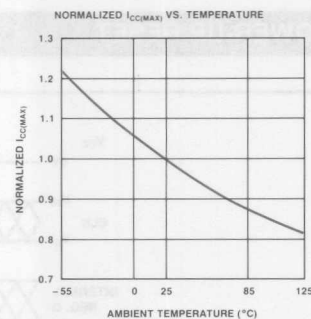
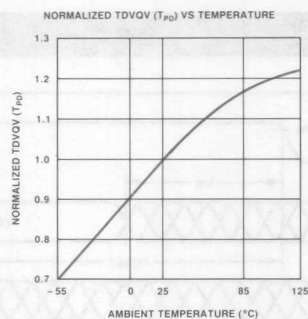
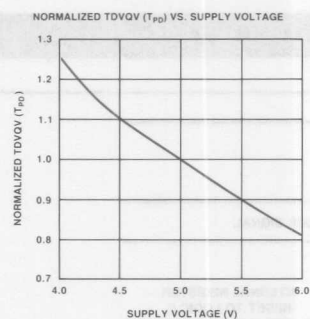
## SOFTWARE DEVELOPMENT TOOLS†

PACKAGE	VENDOR	REVISION
CUPL	Assisted Technology	2.1
ABEL	Data I/O	1.13
PLDtest	Data I/O	†
DASH-ABEL	Data I/O	1.0
PALASM	Monolithic Memories	†

† When emulating PAL devices any revision of the software can be used to create the PAL JEDEC file. The programming hardware will automatically configure the GAL architecture.

Although it is possible to program GAL devices manually, LATTICE strongly recommends the use of approved programming hardware and software. Programming on unapproved equipment will generally void all guarantees.

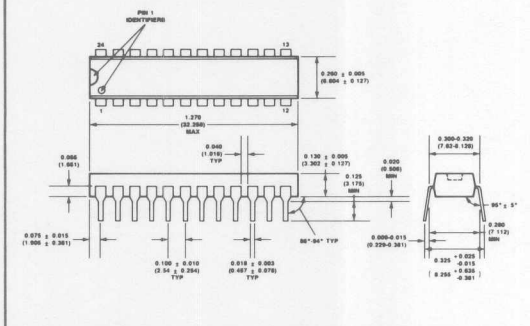
Approved equipment includes LATTICE programming algorithms that program the array, automatically configure the architecture control word, and track the number of program cycles each device has experienced (this information is stored within each GAL device). This in turn assures data retention and reliability. Contact the factory for specific conditions which must be met to gain programming equipment approval or for the current list of approved GAL programming equipment.



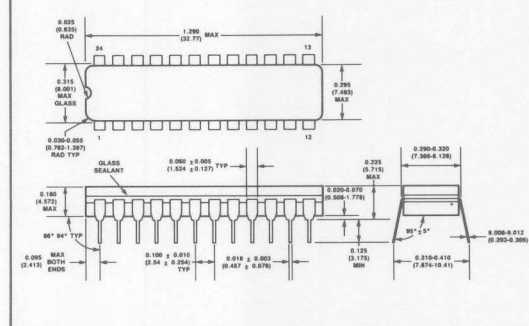


## PACKAGE INFORMATION

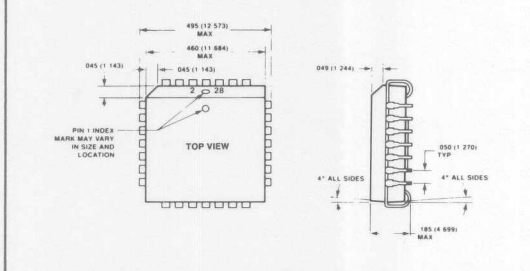
24-PIN, 300-MIL PLASTIC DIP



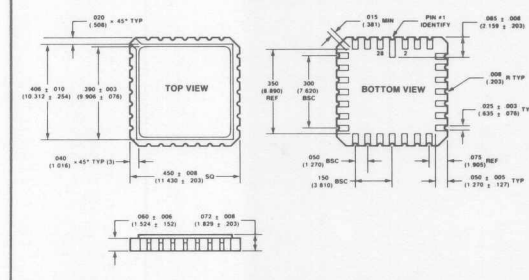
24-PIN, 300-MIL CERDIP



28-PIN PLASTIC LEADED CHIP CARRIER (PLCC)

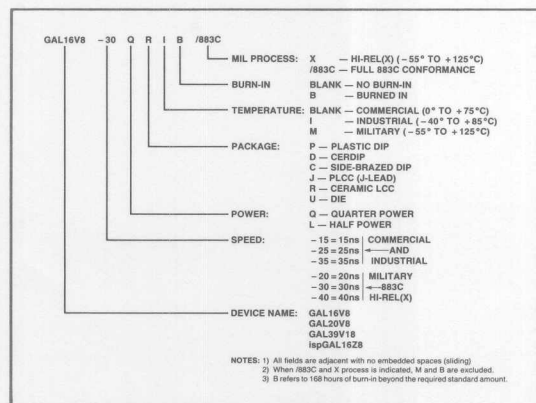


28-PIN LEADLESS CHIP CARRIER (LCC)



NOTE — All dimensions are in inches and parenthetically in millimeters. Inch dimensions govern.

## ORDERING INFORMATION



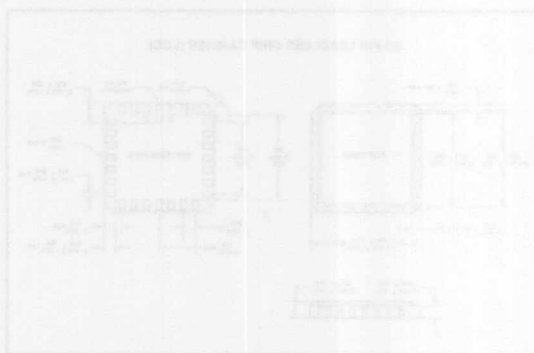
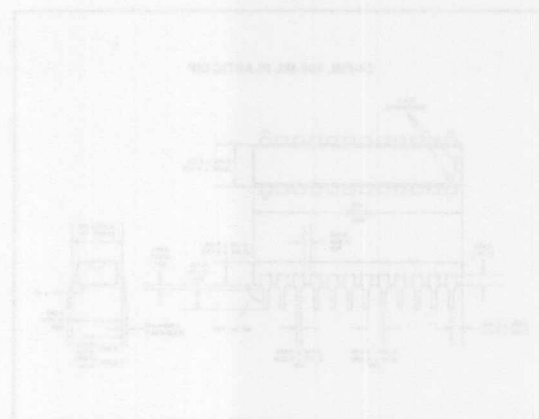
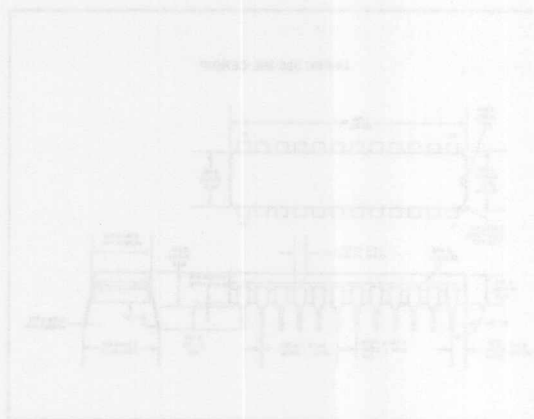
The specifications and information contained herein are subject to change without notice.

## SPEED/POWER CROSS-REFERENCE GUIDE

SPEED	POWER	GAL DEVICE	BIPOLAR PAL DEVICE
15ns	45mA	-15Q	—
15ns	90mA	-15L	—
15ns	180mA	use -15L	B
* 20ns	50mA	-20Q	—
* 20ns	90mA	-20L	—
* 20ns	210mA	use -20L	B MIL
25ns	45mA	-25Q	—
25ns	90mA	-25L	B-2
25ns	180mA	use -25L	A
* 30ns	50mA	-30Q	—
* 30ns	90mA	-30L	B-2 MIL
* 30ns	210mA	use -30L	A MIL
35ns	45mA	-35Q	B-4
35ns	90mA	-35L	A-2
35ns	180mA	use -35L	STD
* 40ns	50mA	-40Q	B-4 MIL
* 40ns	90mA	-40L	A-2 MIL
* 40ns	210mA	use -40L	STD MIL

\* MILITARY TEMPERATURE RANGE

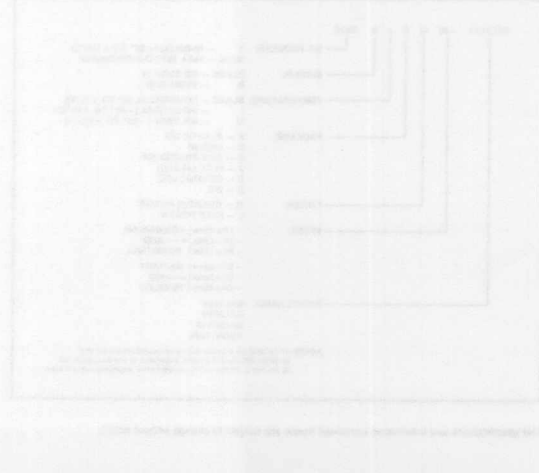
# SECTION 1000 - ELECTRICAL



## SECTION 1000 - ELECTRICAL

TYPE	WINDING	TERMINAL	TERMINAL
1	1000	1000	1000
2	1000	1000	1000
3	1000	1000	1000
4	1000	1000	1000
5	1000	1000	1000
6	1000	1000	1000
7	1000	1000	1000
8	1000	1000	1000
9	1000	1000	1000
10	1000	1000	1000
11	1000	1000	1000
12	1000	1000	1000
13	1000	1000	1000
14	1000	1000	1000
15	1000	1000	1000
16	1000	1000	1000
17	1000	1000	1000
18	1000	1000	1000
19	1000	1000	1000
20	1000	1000	1000
21	1000	1000	1000
22	1000	1000	1000
23	1000	1000	1000
24	1000	1000	1000
25	1000	1000	1000
26	1000	1000	1000
27	1000	1000	1000
28	1000	1000	1000
29	1000	1000	1000
30	1000	1000	1000
31	1000	1000	1000
32	1000	1000	1000
33	1000	1000	1000
34	1000	1000	1000
35	1000	1000	1000
36	1000	1000	1000
37	1000	1000	1000
38	1000	1000	1000
39	1000	1000	1000
40	1000	1000	1000
41	1000	1000	1000
42	1000	1000	1000
43	1000	1000	1000
44	1000	1000	1000
45	1000	1000	1000
46	1000	1000	1000
47	1000	1000	1000
48	1000	1000	1000
49	1000	1000	1000
50	1000	1000	1000
51	1000	1000	1000
52	1000	1000	1000
53	1000	1000	1000
54	1000	1000	1000
55	1000	1000	1000
56	1000	1000	1000
57	1000	1000	1000
58	1000	1000	1000
59	1000	1000	1000
60	1000	1000	1000
61	1000	1000	1000
62	1000	1000	1000
63	1000	1000	1000
64	1000	1000	1000
65	1000	1000	1000
66	1000	1000	1000
67	1000	1000	1000
68	1000	1000	1000
69	1000	1000	1000
70	1000	1000	1000
71	1000	1000	1000
72	1000	1000	1000
73	1000	1000	1000
74	1000	1000	1000
75	1000	1000	1000
76	1000	1000	1000
77	1000	1000	1000
78	1000	1000	1000
79	1000	1000	1000
80	1000	1000	1000
81	1000	1000	1000
82	1000	1000	1000
83	1000	1000	1000
84	1000	1000	1000
85	1000	1000	1000
86	1000	1000	1000
87	1000	1000	1000
88	1000	1000	1000
89	1000	1000	1000
90	1000	1000	1000
91	1000	1000	1000
92	1000	1000	1000
93	1000	1000	1000
94	1000	1000	1000
95	1000	1000	1000
96	1000	1000	1000
97	1000	1000	1000
98	1000	1000	1000
99	1000	1000	1000
100	1000	1000	1000

## SECTION 1000 - ELECTRICAL



### Advance Information

#### FEATURES

- HIGH PERFORMANCE E<sup>2</sup>CMOS TECHNOLOGY
  - 15ns Maximum Propagation Delay
  - $F_{max} = 41.66$  MHz
  - 12ns Maximum from Clock Input to Data Output
  - Output Drive 24ma  $I_{OL}$
  - UltraMOS<sup>®</sup> II Advanced Technology
- 50% REDUCTION IN POWER
  - 90ma Max Active
  - 70ma Max Standby
- E<sup>2</sup> CELL TECHNOLOGY
  - Reconfigurable Logic
  - Reprogrammable Cells
  - 100% Tested/Guaranteed 100% Yields
  - High Speed Electrical Erasure (<50ms),
  - 20 Year Data Retention
- EIGHT OUTPUT LOGIC MACROCELLS
  - Maximum Flexibility for Complex Logic Designs
  - Programmable Output Polarity
  - Also Emulates 20-pin 'B'PAL<sup>®</sup> Devices with Full Function/Fuse Map/Parametric Compatibility
- PRELOAD AND POWER-ON RESET OF ALL REGISTERS
  - 100% Functional Testability
- APPLICATIONS INCLUDE:
  - DMA Control
  - State Machine Control
  - High Speed Video Control
  - Standard Logic Replacement

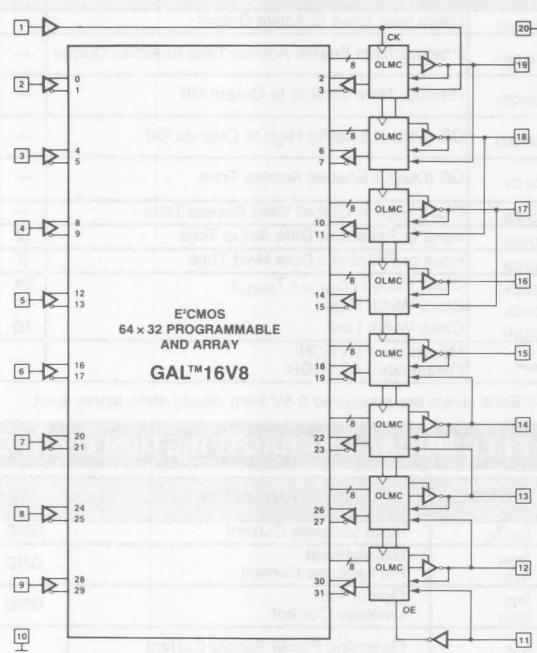
#### DESCRIPTION

The GAL<sup>™</sup> 16V8-15, at 15ns maximum propagation delay time, combines a high performance CMOS process with Electrically Erasable (E<sup>2</sup>) floating gate technology to provide the highest speed performance available in the PLD market. CMOS circuitry allows the GAL 16V8-15 to consume just 90ma maximum  $I_{CC}$  which represents a 50% savings in power when compared to its bipolar counterparts. The E<sup>2</sup> technology offers high speed (50ms) erase times providing the ability to reprogram or reconfigure the devices quickly and efficiently.

The generic architecture provides maximum design flexibility by allowing each Output Logic Macrocell (OLMC) to be configured by the user. An important subset of the many architecture configurations possible with the GAL 16V8 are the PAL<sup>®</sup> architectures listed in the table on the right. The GAL 16V8 is capable of emulating any of these PAL architectures with full function/fuse map/parametric compatibility.

Unique test circuitry and reprogrammable cells allow complete AC, DC, and functional testing during manufacture. Therefore, LATTICE guarantees 100% field programmability and functionality of all GAL products. LATTICE also guarantees 100 erase/write cycles and that data retention exceeds 20 years.

#### FUNCTIONAL BLOCK DIAGRAM



#### GAL<sup>™</sup>/PAL<sup>®</sup> COMPARISON — 15ns DEVICES

PART TYPE	GAL 16V8-15 ARCHITECTURE EMULATION 90ma	PAL 16xxB AVAILABILITY	
		90ma	180ma
16L8	/		/
16H8	/		/
16R8	/		/
16R6	/		/
16R4	/		/
16P8	/		/
16RP8	/		/
16RP6	/		/
16RP4	/		/
10L8	/		/
12L6	/		/
14L4	/		/
16L2	/		/
10H8	/		/
12H6	/		/
14H4	/		/
16H2	/		/
10P8	/		/
12P6	/		/
14P4	/		/
16P2	/		/
16V8	/		/

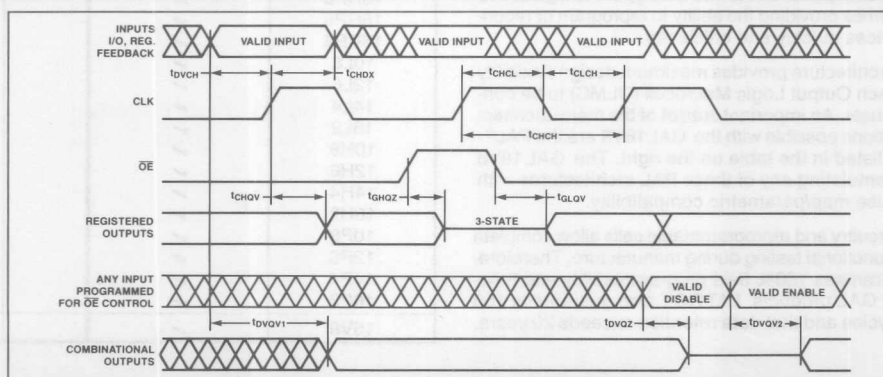
**SWITCHING CHARACTERISTICS OVER OPERATING CONDITIONS**

			TEMPERATURE RANGE								
SYMBOL	PARAMETER		COMMERCIAL				MILITARY		UNITS	TEST CONDITIONS	
			GAL16V8-15		GAL16V8-25		GAL16V8-20			R(Ω)	C <sub>L</sub> (pF)
			MIN.	MAX.	MIN.	MAX.	MIN.	MAX.			
T <sub>DVQV1</sub>	Delay from Input to Active Output		—	15	—	25	—	20	ns	200	50
T <sub>DVQV1</sub>	Product Term Enable Access Time to Active Output		—	15	—	25	—	20	ns	Active High R = ∞ Active Low R = 200	50
T <sub>DVQZ1</sub>	Product Term Disable to Output Off		—	15	—	25	—	20	ns	From V <sub>OH</sub> R = ∞ From V <sub>OL</sub> R = 200	5
T <sub>GHQZ1</sub>	OE (Output Enable) High to Outputs Off		—	15	—	20	—	18	ns	From V <sub>OH</sub> R = ∞ From V <sub>OL</sub> R = 200	5
T <sub>GLQV</sub>	OE (Output Enable) Access Time		—	15	—	20	—	18	ns	Active High R = ∞ From V <sub>OL</sub> R = 200	50
T <sub>CHQV</sub>	Clock High to Output Valid Access Time		—	12	—	15	—	15	ns	200	50
T <sub>DVCH</sub>	Input or Feedback Data Setup Time		12	—	20	—	15	—	ns	—	—
T <sub>CHDX</sub>	Input or Feedback Data Hold Time		0	—	0	—	0	—	ns	—	—
T <sub>CHCH</sub>	Clock Period (T <sub>DVCH</sub> + T <sub>CHQV</sub> )		24	—	35	—	30	—	ns	—	—
T <sub>CHCL</sub>	Clock Width High		10	—	15	—	12	—	ns		
T <sub>CLCH</sub>	Clock Width Low		10	—	15	—	12	—	ns		
f <sub>MAX</sub>	Maximum Frequency	SYNCH. ASYNCH.	—	41.6 66.6	—	28.5 40.0	—	33.3 50.0	MHz	200	50

<sup>1</sup> 3-State levels are measured 0.5V from steady-state active level.

**ELECTRICAL CHARACTERISTICS OVER OPERATING CONDITIONS**
**HALF-POWER  
GAL16V8-15**

SYMBOL	PARAMETER		TEST CONDITIONS	TEMP. RANGE	MIN.	MAX.	UNITS
I <sub>IH</sub> , I <sub>IL</sub>	Input Leakage Current		GND $\leq$ V <sub>IN</sub> $\leq$ V <sub>CC</sub> MAX		—	$\pm 10$	$\mu$ A
I <sub>BZH</sub> , I <sub>BZL</sub>	Bidirectional Pin Leakage Current		GND $\leq$ V <sub>IN</sub> < V <sub>CC</sub> MAX		—	$\pm 10$	$\mu$ A
I <sub>FZL</sub> , I <sub>FZH</sub>	Output Pin Leakage Current		GND $\leq$ V <sub>IN</sub> < V <sub>CC</sub> MAX		—	$\pm 10$	$\mu$ A
I <sub>CC</sub>	Operating Power Supply Current		F = 15 MHz V <sub>CC</sub> = V <sub>CC</sub> MAX	COM'L MIL	— —	90 90	mA
I <sub>OS</sub>	Output Short Circuit Current		V <sub>CC</sub> = 5.0V V <sub>OUT</sub> = GND		—30	—130	mA
I <sub>SB</sub>	Standby Power Supply Current		V <sub>CC</sub> = V <sub>CC</sub> MAX	COM'L MIL	— —	70 70	mA
V <sub>OL</sub>	Output Low Voltage	V <sub>CC</sub> = V <sub>CC</sub> MIN	I <sub>OL</sub> = 24 mA I <sub>OL</sub> = 12 mA	COM'L MIL	— —	0.5 0.5	V
V <sub>OH</sub>	Output High Voltage	V <sub>CC</sub> = V <sub>CC</sub> MIN	I <sub>OH</sub> = —3.2 mA I <sub>OH</sub> = —2.0 mA	COM'L MIL	2.4 2.4	— —	V
V <sub>IH</sub>	Input High Voltage				2.0	V <sub>CC</sub> + 1	V
V <sub>IL</sub>	Input Low Voltage				—	0.8	V

**SWITCHING WAVEFORMS**


NOTE: For detailed technology, logic, timing, programming information and specifications, refer to the GAL16V8 datasheet.

## Advance Information

### FEATURES

- IN-SYSTEM RECONFIGURABLE — 5-VOLT ONLY PROGRAMMING
  - Change Logic "On The Fly"
- HIGH PERFORMANCE E<sup>2</sup> CMOS TECHNOLOGY
  - Low Power: 90 mA Max Active  
70 mA Max Standby
  - High Speed: 25 ns Access Max  
35 ns Access Max
- EIGHT OUTPUT LOGIC MACROCELLS
  - Maximum Flexibility for Complex Logic Designs
  - Also Emulates 20-pin PAL® Devices with Full Function/Fuse Map/Parametric Compatibility
- PRELOAD AND POWER-ON RESET OF ALL REGISTERS
  - 100% Functional Testability
- 24 PIN 0.3 INCH NARROW DIP SAVES SPACE
- MINIMUM 10,000 ERASE/WRITE CYCLES
- DATA RETENTION EXCEEDS 20 YEARS
- APPLICATIONS INCLUDE:
  - Multi-mode System Operation
  - Configurable Memory Mapping
  - Update Systems with Firmware Instead of Hardware

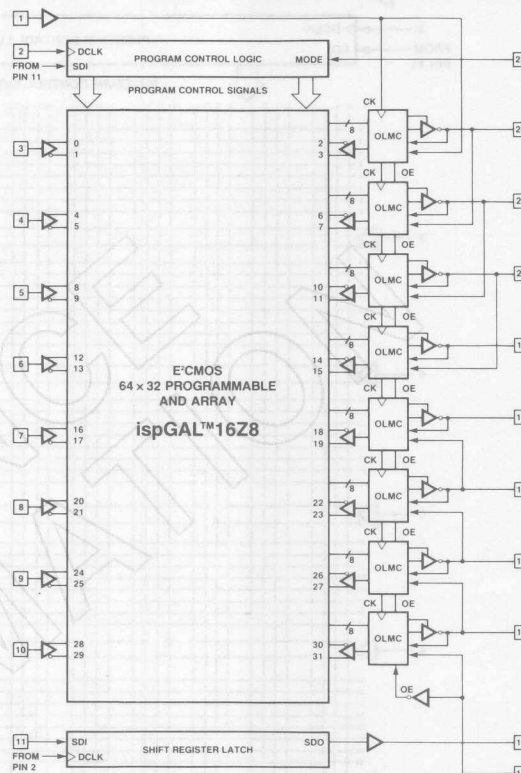
### DESCRIPTION

The LATTICE ispGAL16Z8 (patent pending) is a revolutionary programmable logic device featuring a 5-volt only in-system reprogrammability. The device combines a high performance CMOS process with Electrically Erasable (E<sup>2</sup>) floating gate technology. This programmable memory technology applied to array logic provides designers with reconfigurable logic and bipolar performance at significantly reduced power levels.

The 24-pin ispGAL16Z8 is architecturally equivalent to the familiar 20-pin GAL™16V8 but includes 4 extra pins to control in-system programming. It features 8 programmable Output Logic Macrocells (OLMC) allowing each output to be configured by the user. Additionally, the ispGAL16Z8 is capable of emulating, in a functional/fuse map compatible device, all common 20-pin PAL device architectures.

Unique test circuitry and reprogrammable fuses allow complete AC, DC, fuse and functionality testing during manufacture. Therefore, LATTICE guarantees 100% field programmability and functionality of the ispGAL devices. In addition, electronic signature is available to provide positive device ID. A security circuit is built-in, providing proprietary designs with copy protection.

### FUNCTIONAL BLOCK DIAGRAM



### PIN NAMES

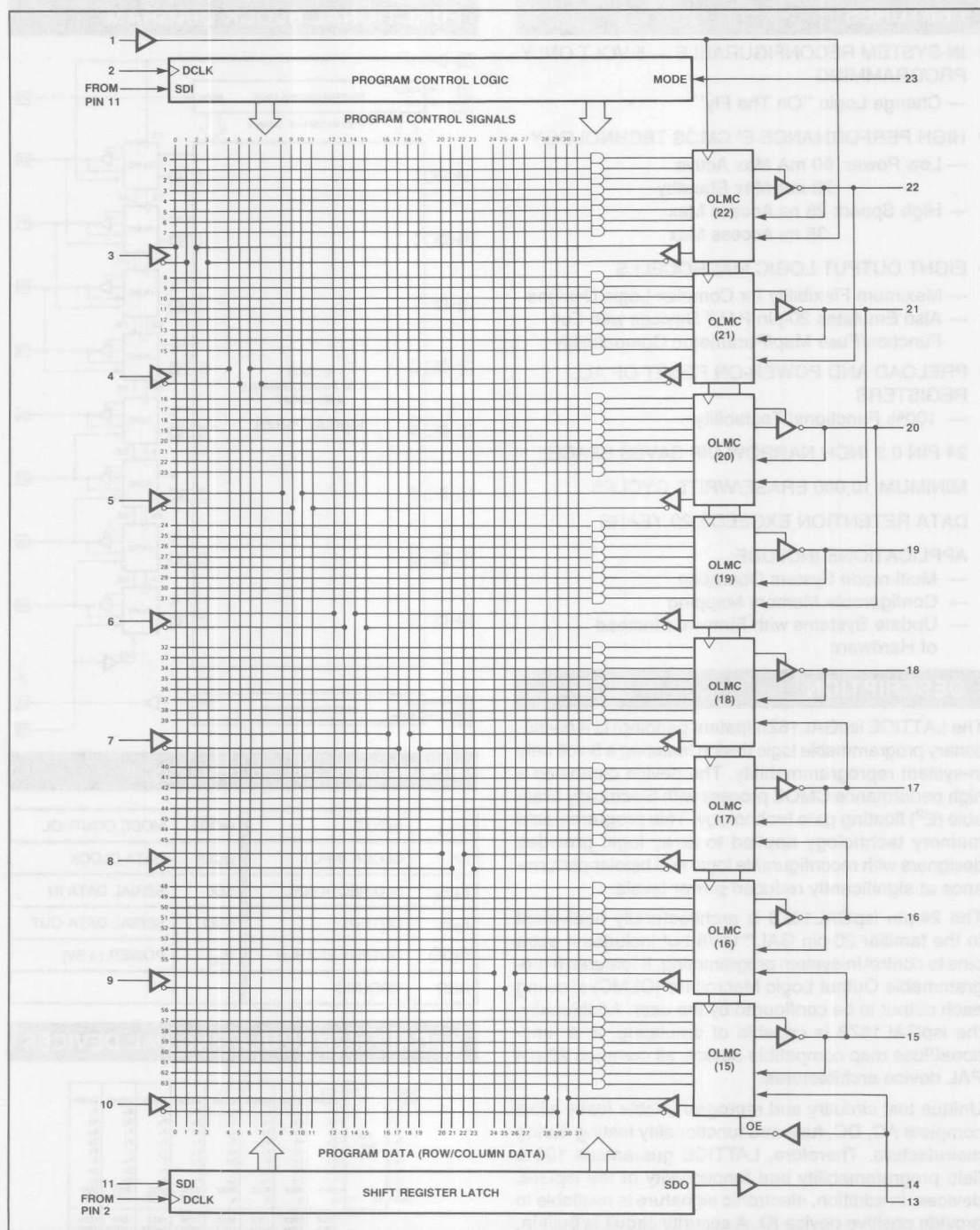
I <sub>0</sub> -I <sub>15</sub>	INPUT	MODE	MODE CONTROL
CK	CLOCK INPUT	DCLK	DATA CLOCK
B <sub>0</sub> -B <sub>5</sub>	BI-DIRECTIONAL	SDI	SERIAL DATA IN
F <sub>0</sub> -F <sub>7</sub>	OUTPUT	SDO	SERIAL DATA OUT
$\bar{G}(\bar{OE})$	OUTPUT ENABLE	V <sub>CC</sub>	POWER (+ 5V)
GND	GROUND		

### ispGAL™16Z8 EMULATING PAL DEVICES

1	24	VCC	MODE	MODE	MODE	MODE	MODE	MODE	MODE
DCLK	I <sub>0</sub>	F <sub>7</sub>	I <sub>11</sub>	I <sub>13</sub>	I <sub>15</sub>	F <sub>7</sub>	B <sub>1</sub>	B <sub>3</sub>	F <sub>1</sub>
I <sub>1</sub>	I <sub>2</sub>	F <sub>6</sub>	F <sub>5</sub>	I <sub>12</sub>	I <sub>14</sub>	F <sub>6</sub>	F <sub>5</sub>	B <sub>2</sub>	B <sub>5</sub>
I <sub>3</sub>	I <sub>4</sub>	F <sub>4</sub>	F <sub>3</sub>	F <sub>2</sub>	F <sub>1</sub>	F <sub>4</sub>	F <sub>3</sub>	F <sub>2</sub>	B <sub>4</sub>
I <sub>5</sub>	I <sub>6</sub>	F <sub>3</sub>	F <sub>2</sub>	F <sub>1</sub>	F <sub>0</sub>	F <sub>3</sub>	F <sub>2</sub>	F <sub>1</sub>	B <sub>3</sub>
I <sub>7</sub>	I <sub>8</sub>	F <sub>2</sub>	F <sub>1</sub>	F <sub>0</sub>	I <sub>12</sub>	F <sub>2</sub>	F <sub>1</sub>	B <sub>2</sub>	B <sub>2</sub>
I <sub>9</sub>	I <sub>10</sub>	F <sub>1</sub>	F <sub>0</sub>	I <sub>11</sub>	I <sub>13</sub>	F <sub>1</sub>	F <sub>0</sub>	B <sub>1</sub>	B <sub>1</sub>
I <sub>11</sub>	I <sub>12</sub>	F <sub>0</sub>	I <sub>10</sub>	I <sub>9</sub>	I <sub>8</sub>	F <sub>0</sub>	B <sub>0</sub>	B <sub>0</sub>	F <sub>0</sub>
SDI	SDO	I <sub>10</sub>	SDO	I <sub>9</sub>	I <sub>8</sub>	SDO	SDO	SDO	SDO
GND	I <sub>13</sub>	I <sub>14</sub>	I <sub>15</sub>	I <sub>16</sub>	I <sub>17</sub>	I <sub>18</sub>	I <sub>19</sub>	I <sub>20</sub>	I <sub>21</sub>
		I <sub>22</sub>	I <sub>23</sub>	I <sub>24</sub>	I <sub>25</sub>	I <sub>26</sub>	I <sub>27</sub>	I <sub>28</sub>	I <sub>29</sub>
		I <sub>30</sub>	I <sub>31</sub>	I <sub>32</sub>	I <sub>33</sub>	I <sub>34</sub>	I <sub>35</sub>	I <sub>36</sub>	I <sub>37</sub>
		I <sub>38</sub>	I <sub>39</sub>	I <sub>40</sub>	I <sub>41</sub>	I <sub>42</sub>	I <sub>43</sub>	I <sub>44</sub>	I <sub>45</sub>
		I <sub>46</sub>	I <sub>47</sub>	I <sub>48</sub>	I <sub>49</sub>	I <sub>50</sub>	I <sub>51</sub>	I <sub>52</sub>	I <sub>53</sub>
		I <sub>54</sub>	I <sub>55</sub>	I <sub>56</sub>	I <sub>57</sub>	I <sub>58</sub>	I <sub>59</sub>	I <sub>60</sub>	I <sub>61</sub>
		I <sub>62</sub>	I <sub>63</sub>	I <sub>64</sub>	I <sub>65</sub>	I <sub>66</sub>	I <sub>67</sub>	I <sub>68</sub>	I <sub>69</sub>
		I <sub>70</sub>	I <sub>71</sub>	I <sub>72</sub>	I <sub>73</sub>	I <sub>74</sub>	I <sub>75</sub>	I <sub>76</sub>	I <sub>77</sub>
		I <sub>78</sub>	I <sub>79</sub>	I <sub>80</sub>	I <sub>81</sub>	I <sub>82</sub>	I <sub>83</sub>	I <sub>84</sub>	I <sub>85</sub>
		I <sub>86</sub>	I <sub>87</sub>	I <sub>88</sub>	I <sub>89</sub>	I <sub>90</sub>	I <sub>91</sub>	I <sub>92</sub>	I <sub>93</sub>
		I <sub>94</sub>	I <sub>95</sub>	I <sub>96</sub>	I <sub>97</sub>	I <sub>98</sub>	I <sub>99</sub>	I <sub>100</sub>	I <sub>101</sub>
		I <sub>102</sub>	I <sub>103</sub>	I <sub>104</sub>	I <sub>105</sub>	I <sub>106</sub>	I <sub>107</sub>	I <sub>108</sub>	I <sub>109</sub>
		I <sub>110</sub>	I <sub>111</sub>	I <sub>112</sub>	I <sub>113</sub>	I <sub>114</sub>	I <sub>115</sub>	I <sub>116</sub>	I <sub>117</sub>
		I <sub>118</sub>	I <sub>119</sub>	I <sub>120</sub>	I <sub>121</sub>	I <sub>122</sub>	I <sub>123</sub>	I <sub>124</sub>	I <sub>125</sub>
		I <sub>126</sub>	I <sub>127</sub>	I <sub>128</sub>	I <sub>129</sub>	I <sub>130</sub>	I <sub>131</sub>	I <sub>132</sub>	I <sub>133</sub>
		I <sub>134</sub>	I <sub>135</sub>	I <sub>136</sub>	I <sub>137</sub>	I <sub>138</sub>	I <sub>139</sub>	I <sub>140</sub>	I <sub>141</sub>
		I <sub>142</sub>	I <sub>143</sub>	I <sub>144</sub>	I <sub>145</sub>	I <sub>146</sub>	I <sub>147</sub>	I <sub>148</sub>	I <sub>149</sub>
		I <sub>150</sub>	I <sub>151</sub>	I <sub>152</sub>	I <sub>153</sub>	I <sub>154</sub>	I <sub>155</sub>	I <sub>156</sub>	I <sub>157</sub>
		I <sub>158</sub>	I <sub>159</sub>	I <sub>160</sub>	I <sub>161</sub>	I <sub>162</sub>	I <sub>163</sub>	I <sub>164</sub>	I <sub>165</sub>
		I <sub>166</sub>	I <sub>167</sub>	I <sub>168</sub>	I <sub>169</sub>	I <sub>170</sub>	I <sub>171</sub>	I <sub>172</sub>	I <sub>173</sub>
		I <sub>174</sub>	I <sub>175</sub>	I <sub>176</sub>	I <sub>177</sub>	I <sub>178</sub>	I <sub>179</sub>	I <sub>180</sub>	I <sub>181</sub>
		I <sub>182</sub>	I <sub>183</sub>	I <sub>184</sub>	I <sub>185</sub>	I <sub>186</sub>	I <sub>187</sub>	I <sub>188</sub>	I <sub>189</sub>
		I <sub>190</sub>	I <sub>191</sub>	I <sub>192</sub>	I <sub>193</sub>	I <sub>194</sub>	I <sub>195</sub>	I <sub>196</sub>	I <sub>197</sub>
		I <sub>198</sub>	I <sub>199</sub>	I <sub>200</sub>	I <sub>201</sub>	I <sub>202</sub>	I <sub>203</sub>	I <sub>204</sub>	I <sub>205</sub>
		I <sub>206</sub>	I <sub>207</sub>	I <sub>208</sub>	I <sub>209</sub>	I <sub>210</sub>	I <sub>211</sub>	I <sub>212</sub>	I <sub>213</sub>
		I <sub>214</sub>	I <sub>215</sub>	I <sub>216</sub>	I <sub>217</sub>	I <sub>218</sub>	I <sub>219</sub>	I <sub>220</sub>	I <sub>221</sub>
		I <sub>222</sub>	I <sub>223</sub>	I <sub>224</sub>	I <sub>225</sub>	I <sub>226</sub>	I <sub>227</sub>	I <sub>228</sub>	I <sub>229</sub>
		I <sub>230</sub>	I <sub>231</sub>	I <sub>232</sub>	I <sub>233</sub>	I <sub>234</sub>	I <sub>235</sub>	I <sub>236</sub>	I <sub>237</sub>
		I <sub>238</sub>	I <sub>239</sub>	I <sub>240</sub>	I <sub>241</sub>	I <sub>242</sub>	I <sub>243</sub>	I <sub>244</sub>	I <sub>245</sub>
		I <sub>246</sub>	I <sub>247</sub>	I <sub>248</sub>	I <sub>249</sub>	I <sub>250</sub>	I <sub>251</sub>	I <sub>252</sub>	I <sub>253</sub>
		I <sub>254</sub>	I <sub>255</sub>	I <sub>256</sub>	I <sub>257</sub>	I <sub>258</sub>	I <sub>259</sub>	I <sub>260</sub>	I <sub>261</sub>
		I <sub>262</sub>	I <sub>263</sub>	I <sub>264</sub>	I <sub>265</sub>	I <sub>266</sub>	I <sub>267</sub>	I <sub>268</sub>	I <sub>269</sub>
		I <sub>270</sub>	I <sub>271</sub>	I <sub>272</sub>	I <sub>273</sub>	I <sub>274</sub>	I <sub>275</sub>	I <sub>276</sub>	I <sub>277</sub>
		I <sub>278</sub>	I <sub>279</sub>	I <sub>280</sub>	I <sub>281</sub>	I <sub>282</sub>	I <sub>283</sub>	I <sub>284</sub>	I <sub>285</sub>
		I <sub>286</sub>	I <sub>287</sub>	I <sub>288</sub>	I <sub>289</sub>	I <sub>290</sub>	I <sub>291</sub>	I <sub>292</sub>	I <sub>293</sub>
		I <sub>294</sub>	I <sub>295</sub>	I <sub>296</sub>	I <sub>297</sub>	I <sub>298</sub>	I <sub>299</sub>	I <sub>300</sub>	I <sub>301</sub>
		I <sub>302</sub>	I <sub>303</sub>	I <sub>304</sub>	I <sub>305</sub>	I <sub>306</sub>	I <sub>307</sub>	I <sub>308</sub>	I <sub>309</sub>
		I <sub>310</sub>	I <sub>311</sub>	I <sub>312</sub>	I <sub>313</sub>	I <sub>314</sub>	I <sub>315</sub>	I <sub>316</sub>	I <sub>317</sub>
		I <sub>318</sub>	I <sub>319</sub>	I <sub>320</sub>	I <sub>321</sub>	I <sub>322</sub>	I <sub>323</sub>	I <sub>324</sub>	I <sub>325</sub>
		I <sub>326</sub>	I <sub>327</sub>	I <sub>328</sub>	I <sub>329</sub>	I <sub>330</sub>	I <sub>331</sub>	I <sub>332</sub>	I <sub>333</sub>
		I <sub>334</sub>	I <sub>335</sub>	I <sub>336</sub>	I <sub>337</sub>	I <sub>338</sub>	I <sub>339</sub>	I <sub>340</sub>	I <sub>341</sub>
		I <sub>342</sub>	I <sub>343</sub>	I <sub>344</sub>	I <sub>345</sub>	I <sub>346</sub>	I <sub>347</sub>	I <sub>348</sub>	I <sub>349</sub>
		I <sub>350</sub>	I <sub>351</sub>	I <sub>352</sub>	I <sub>353</sub>	I <sub>354</sub>	I <sub>355</sub>	I <sub>356</sub>	I <sub>357</sub>
		I <sub>358</sub>	I <sub>359</sub>	I <sub>360</sub>	I <sub>361</sub>	I <sub>362</sub>	I <sub>363</sub>	I <sub>364</sub>	I <sub>365</sub>
		I <sub>366</sub>	I <sub>367</sub>	I <sub>368</sub>	I <sub>369</sub>	I <sub>370</sub>	I <sub>371</sub>	I <sub>372</sub>	I <sub>373</sub>
		I <sub>374</sub>	I <sub>375</sub>	I <sub>376</sub>	I <sub>377</sub>	I <sub>378</sub>	I <sub>379</sub>	I <sub>380</sub>	I <sub>381</sub>
		I <sub>382</sub>	I <sub>383</sub>	I <sub>384</sub>	I <sub>385</sub>	I <sub>386</sub>	I <sub>387</sub>	I <sub>388</sub>	I <sub>389</sub>
		I <sub>390</sub>	I <sub>391</sub>	I <sub>392</sub>	I <sub>393</sub>	I <sub>394</sub>	I <sub>395</sub>	I <sub>396</sub>	I <sub>397</sub>
		I <sub>398</sub>	I <sub>399</sub>	I <sub>400</sub>	I <sub>401</sub>	I <sub>402</sub>	I <sub>403</sub>	I <sub>404</sub>	I <sub>405</sub>
		I <sub>406</sub>	I <sub>407</sub>	I <sub>408</sub>	I <sub>409</sub>	I <sub>410</sub>	I <sub>411</sub>	I <sub>412</sub>	I <sub>413</sub>
		I <sub>414</sub>	I <sub>415</sub>	I <sub>416</sub>	I <sub>417</sub>	I <sub>418</sub>	I <sub>419</sub>	I <sub>420</sub>	I <sub>421</sub>
		I <sub>422</sub>	I <sub>423</sub>	I <sub>424</sub>	I <sub>425</sub>	I <sub>426</sub>	I <sub>427</sub>	I <sub>428</sub>	I <sub>429</sub>
		I <sub>430</sub>	I <sub>431</sub>	I <sub>432</sub>	I <sub>433</sub>	I <sub>434</sub>	I <sub>435</sub>	I <sub>436</sub>	I <sub>437</sub>
		I <sub>438</sub>	I <sub>439</sub>	I <sub>440</sub>	I <sub>441</sub>	I <sub>442</sub>	I <sub>443</sub>	I <sub>444</sub>	I <sub>445</sub>
		I <sub>446</sub>	I <sub>447</sub>	I <sub>448</sub>	I <sub>449</sub>	I <sub>450</sub>	I <sub>451</sub>	I <sub>452</sub>	I <sub>453</sub>
		I <sub>454</sub>	I <sub>455</sub>	I <sub>456</sub>	I <sub>457</sub>	I <sub>458</sub>	I <sub>459</sub>	I <sub>460</sub>	I <sub>461</sub>
		I <sub>462</sub>	I <sub>463</sub>	I <sub>464</sub>	I <sub>465</sub>	I <sub>466</sub>	I <sub>467</sub>	I <sub>468</sub>	I <sub>469</sub>
		I <sub>470</sub>	I <sub>471</sub>	I <sub>472</sub>	I <sub>473</sub>	I <sub>474</sub>	I <sub>475</sub>	I <sub>476</sub>	I <sub>477</sub>
		I <sub>478</sub>	I <sub>479</sub>	I <sub>480</sub>	I <sub>481</sub>	I <sub>482</sub>	I <sub>483</sub>	I <sub>484</sub>	I <sub>485</sub>
		I <sub>486</sub>	I <sub>487</sub>	I <sub>488</sub>	I <sub>489</sub>	I <sub>490</sub>	I <sub>491</sub>	I <sub>492</sub>	I <sub>493</sub>
		I <sub>494</sub>	I <sub>495</sub>	I <sub>496</sub>	I <sub>497</sub>	I <sub>498</sub>	I <sub>499</sub>	I <sub>500</sub>	I <sub>501</sub>
		I <sub>502</sub>	I <sub>503</sub>	I <sub>504</sub>	I <sub>505</sub>	I <sub>506</sub>	I <sub>507</sub>	I <sub>508</sub>	I <sub>509</sub>
		I <sub>510</sub>	I <sub>511</sub>	I <sub>512</sub>	I <sub>513</sub>	I <sub>514</sub>	I <sub>515</sub>	I <sub>516</sub>	I <sub>517</sub>
		I <sub>518</sub>	I <sub>519</sub>	I <sub>520</sub>	I <sub>521</sub>	I <sub>522</sub>	I <sub>523</sub>	I <sub>524</sub>	I <sub>525</sub>
		I <sub>526</sub>	I <sub>527</sub>	I <sub>528</sub>	I <sub>529</sub>	I <sub>530</sub>	I <sub>531</sub>	I <sub>532</sub>	I <sub>533</sub>
		I <sub>534</sub>	I <sub>535</sub>	I <sub>536</sub>	I <sub>537</sub>	I <sub>538</sub>	I <sub>539</sub>	I <sub>540</sub>	I <sub>541</sub>
		I <sub>542</sub>	I <sub>543</sub>	I <sub>544</sub>	I <sub>545</sub>	I <sub>546</sub>	I <sub>547</sub>	I <sub>548</sub>	I <sub>549</sub>
		I <sub>550</sub>	I <sub>551</sub>	I <sub>552</sub>	I <sub>553</sub>	I <sub>554</sub>	I <sub>555</sub>	I <sub>556</sub>	I <sub>557</sub>
		I <sub>558</sub>	I <sub>559</sub>	I <sub>560</sub>	I <sub>561</sub>	I <sub>562</sub>	I <sub>563</sub>	I <sub>564</sub>	I <sub>565</sub>
		I <sub>566</sub>	I <sub>567</sub>	I <sub>568</sub>	I <sub>569</sub>	I <sub>570</sub>	I <sub>571</sub>	I <sub>572</sub>	I <sub>573</sub>
		I <sub>574</sub>	I <sub>575</sub>	I <sub>576</sub>	I <sub>577</sub>	I <sub>578</sub>	I <sub>579</sub>	I <sub>580</sub>	I <sub>581</sub>
		I <sub>582</sub>	I <sub>583</sub>	I <sub>584</sub>	I <sub>585</sub>	I <sub>586</sub>	I <sub>587</sub>	I <sub>588</sub>	I <sub>589</sub>
		I <sub>590</sub>	I <sub>591</sub>	I <sub>592</sub>	I <sub>593</sub>	I <sub>594</sub>	I <sub>595</sub>	I <sub>596</sub>	I <sub>597</sub>
		I <sub>598</sub>	I <sub>599</sub>	I <sub>600</sub>	I <sub>601</sub>	I <sub>602</sub>	I <sub>603</sub>	I <sub>604</sub>	I <sub>605</sub>
		I <sub>606</sub>	I <sub>607</sub>	I <sub>608</sub>	I <sub>609</sub>	I <sub>610</sub>	I <sub>611</sub>	I <sub>612</sub>	I <sub>613</sub>
		I <sub>614</sub>	I <sub>615</sub>	I <sub>616</sub>	I <sub>617</sub>	I <sub>618</sub>	I <sub>619</sub>	I <sub>620</sub>	I <sub>621</sub>
		I <sub>622</sub>	I <sub>623</sub>	I <sub>624</sub>	I <sub>625</sub>	I <sub>626</sub>	I <sub>627</sub>	I <sub>628</sub>	I <sub>629</sub>
		I <sub>630</sub>	I <sub>631</sub>	I <sub>632</sub>	I <sub>633</sub>	I <sub>634</sub>	I <sub>635</sub>	I <sub>636</sub>	I <sub>637</sub>
		I <sub>638</sub>	I <sub>639</sub>	I <sub>640</sub>	I <sub>641</sub>	I <sub>642</sub>	I <sub>643</sub>	I <sub>644</sub>	I <sub>645</sub>
		I <sub>646</sub>	I <sub>647</sub>	I <sub>648</sub>	I <sub>649</sub>	I <sub>650</sub>	I <sub>651</sub>	I <sub>652</sub>	I <sub>653</sub>
		I <sub>654</sub>	I <sub>655</sub>	I <sub>656</sub>	I <sub>657</sub>	I <sub>658</sub>	I <sub>659</sub>	I <sub>660</sub>	I <sub>661</sub>
		I <sub>662</sub>	I <sub>663</sub>	I <sub>664</sub>	I <sub>665</sub>	I <sub>666</sub>	I <sub>667</sub>	I <sub>668</sub>	I <sub>669</sub>
		I <sub>670</sub>	I <sub>671</sub>	I <sub>672</sub>	I <sub>673</sub>	I <sub>674</sub>	I <sub>675</sub>	I <sub>676</sub>	I



## ispGAL16Z8 LOGIC DIAGRAM



## Advance Information

### FEATURES

- ELECTRICALLY ERASABLE CELL TECHNOLOGY
  - Instantly Reconfigurable Logic
  - Instantly Reprogrammable Cells
  - Guaranteed 100% Yields
- HIGH PERFORMANCE E<sup>2</sup>CMOS TECHNOLOGY
  - High Speed: 30ns Max. Propagation Delay
  - 20ns Max. Setup Time
- UNPRECEDENTED FUNCTIONAL DENSITY
  - 10 Output Logic Macro Cells
  - 8 State Logic Macro Cells
  - 20 Input Logic Macro Cells
- HIGH-LEVEL DESIGN FLEXIBILITY
  - 39 Array Inputs, 10 Outputs
  - Separate State Register and Input Clock Pins
  - Supersets Existing 24 pin PAL<sup>®</sup> and IFL devices
  - FPLA Architecture
- SPACE-SAVING 24-PIN, 300-MIL DIP
- HIGH-SPEED PROGRAMMING ALGORITHM
- 20-YEAR DATA RETENTION

### DESCRIPTION

Using a high performance E<sup>2</sup>CMOS technology, Lattice Semiconductor has produced a next-generation programmable logic device, the GAL39V18. Using an FPLA architecture known for its superior flexibility in state-machine design, the GAL39V18 offers the highest degree of functional integration and flexibility currently available in a 24-pin, 300-mil package.

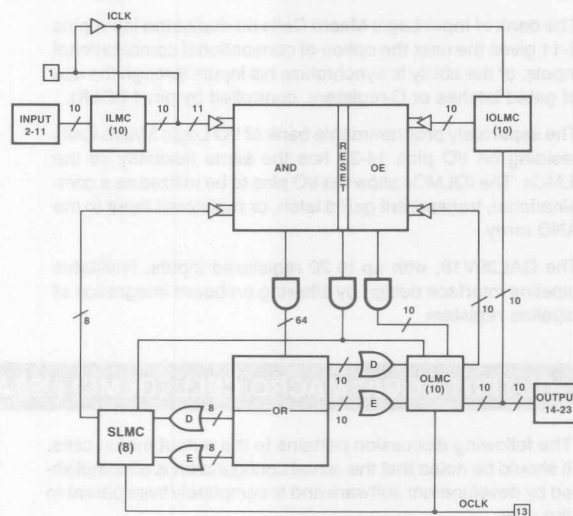
The GAL39V18 has 10 programmable Output Logic Macro Cells (OLMCs) and 8 Programmable "Buried" State Logic Macro Cells (SLMCs). In addition, there are 20 Input Logic Macro Cells (ILMCs). Two Clock inputs are provided for independent control of the Input and State Macro Cells.

Advanced features that simplify programming and reduce test time, coupled with E<sup>2</sup>CMOS reprogrammable cells, enable complete AC, DC, programmability, and functionality test of each GAL39V18 during manufacture. This allows Lattice to guarantee 100% field programmability and functionality to data-sheet specifications.

Programming is accomplished using standard hardware and software tools. Lattice guarantees a minimum of 100 erase/write cycles, and data retention to exceed 20 years. An Electronic Signature word has been provided for user-defined data. In addition, a security cell is available to protect proprietary designs.

© Copyright May 1986

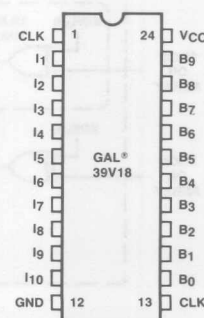
### FUNCTIONAL BLOCK DIAGRAM



### PIN NAMES

I <sub>0</sub> -I <sub>10</sub>	INPUT	B <sub>0</sub> -B <sub>9</sub>	BIDIRECTIONAL
ICLK	INPUT CLOCK	V <sub>CC</sub>	POWER (+5)
OCLK	OUTPUT CLOCK	GND	GROUND

### PIN CONFIGURATION



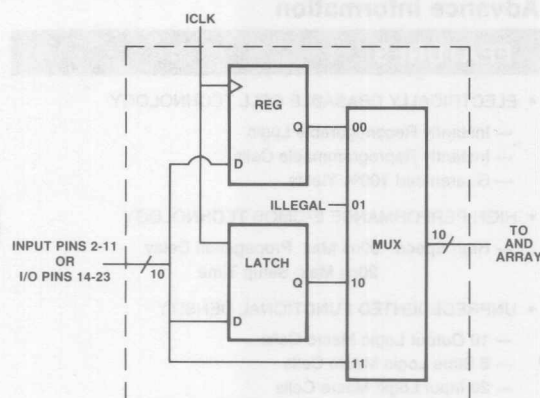
## INPUT LOGIC MACRO CELL (ILMC) AND I/O LOGIC MACRO CELL (IOLMC)

The following discussion pertains to the input macro cells. It should be noted that the actual configuration is accomplished by development software and is completely transparent to the user.

The bank of Input Logic Macro Cells on dedicated input pins 2-11 gives the user the option of conventional combinational inputs, or the ability to synchronize his inputs through the use of gated latches or D-registers, controlled by pin 1 (ICLK).

The separately programmable bank of I/O Logic Macro Cells residing on I/O pins 14-23 has the same flexibility as the ILMCs. The IOLMCs allow the I/O pins to be utilized as a combinational, transparent gated latch, or registered input to the AND array.

The GAL39V18, with up to 20 registered inputs, facilitates pipeline interface design by allowing on-board integration of pipeline registers.



## OUTPUT LOGIC MACRO CELL (OLMC) / STATE LOGIC MACRO CELL (SLMC)

The following discussion pertains to the output macro cells. It should be noted that the actual configuration is accomplished by development software and is completely transparent to the user.

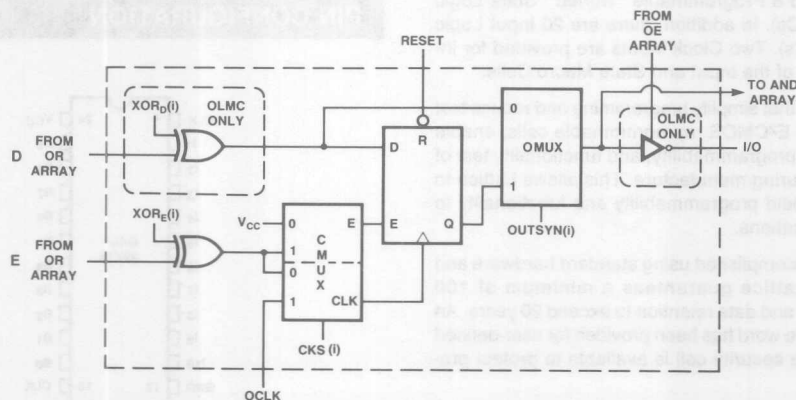
The key to the power of the GAL39V18 is the availability of 18 programmable macro cells for building large state machines. The 8 "buried" SLMCs provide dedicated state feedback to the AND array, while the 10 OLMCs serve both as dedicated-state feedback and device outputs with individual output-enable control, which enables the register to be used in a "buried" manner, while still allowing the I/O pin to be used as a separate input.

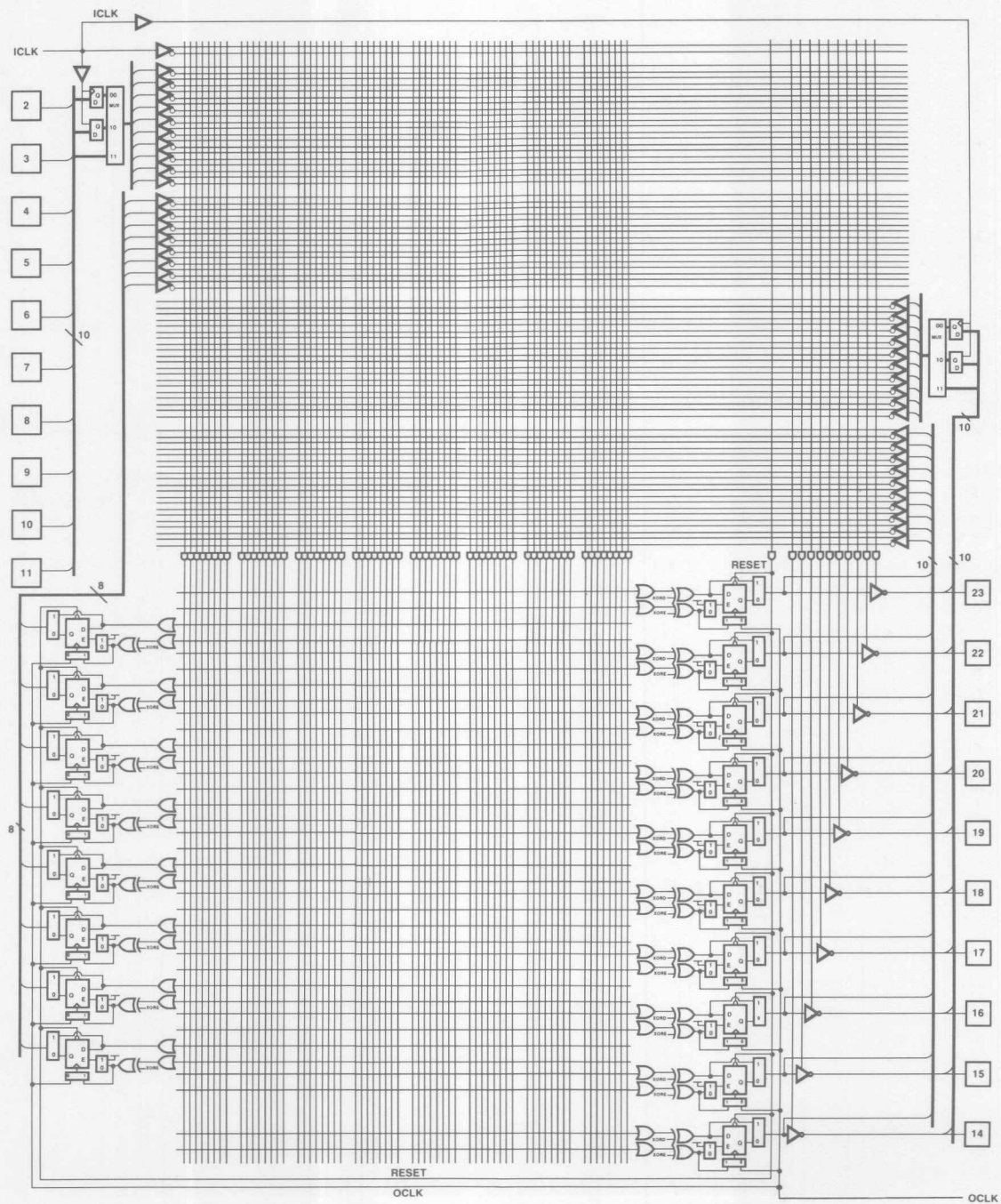
The OLMCs are built of two 64-input sum terms (D and E) controlling a programmable block that can be configured in one of three basic modes: Combinational, D Register with Clock Enable, and D Register with Programmable Clock. In the first mode, the Combinational mode, the E sum-term is not

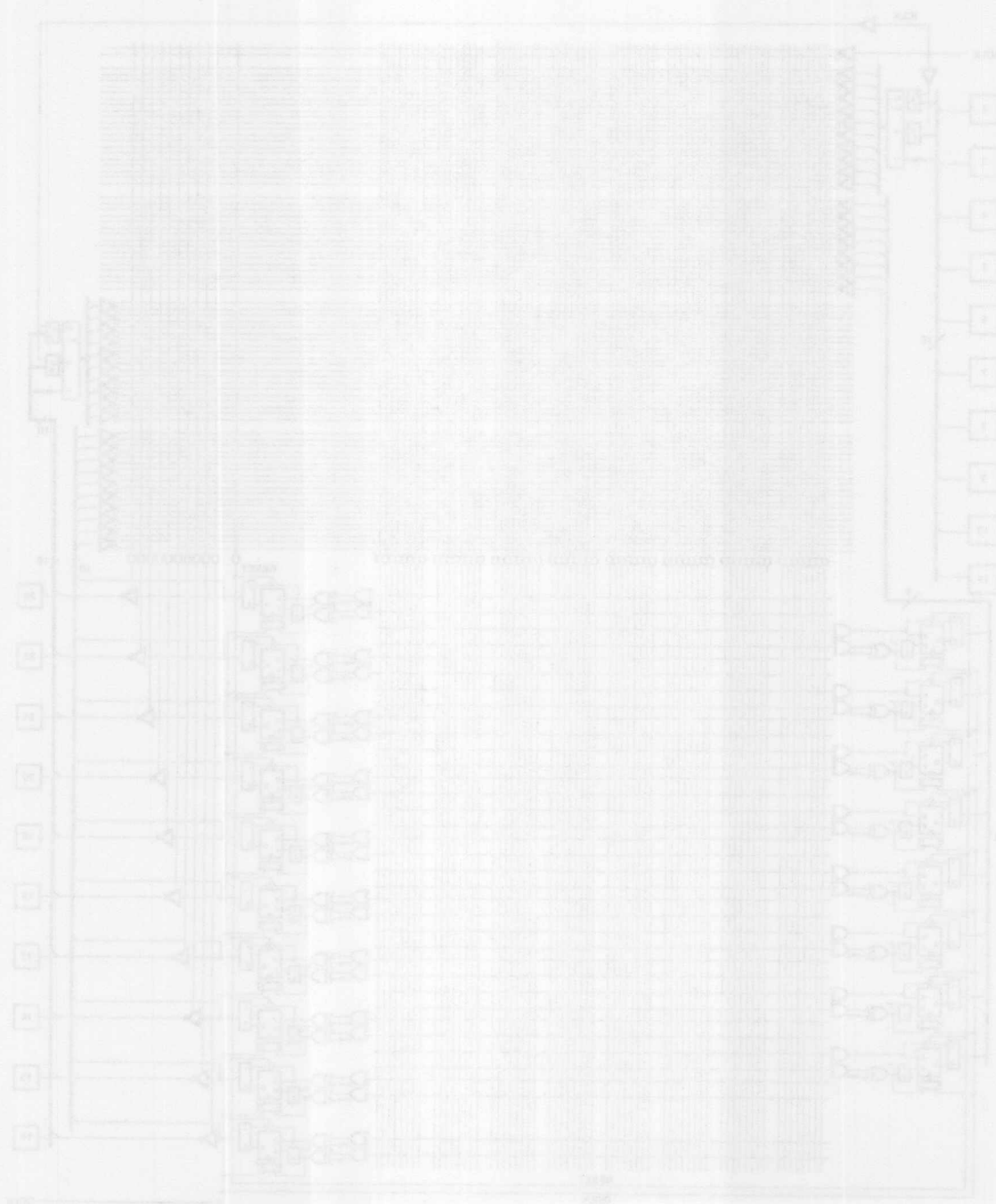
used and the D sum-term data asynchronously routes to the AND feedback and to the device outputs under Output Enable control. In the second mode, the D and E register mode, the D register is clocked by pin 13 (OCLK), which is enabled on an output-by-output basis by the E sum term. This allows the ability to "hold state." In the third mode, the D register is clocked by the E sum term. This programmable clock feature allows another way to "hold state," and the ability to use multiple user-defined clocks. Both the D and E sum terms have programmable polarity to eliminate off-board inverters, allow choice of programmable clock polarity, and to allow maximum logic-design flexibility through DeMorgan's theorem.

The SLMC's are identical to the OLMCs with one exception. There is no need for a D sum-term polarity control, since the AND array feedback provides True/Complement flexibility.

A special State Register Preload Mode allows control of all state feedback to the array for optimum testability.



GAL<sup>®</sup> 39V18 LOGIC DIAGRAM



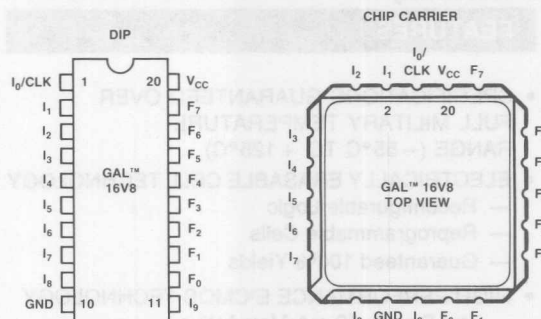




**ABSOLUTE MAXIMUM RATINGS<sup>(1)</sup>**

Supply voltage  $V_{CC}$  ..... - .5 to +7V  
 Input voltage applied ..... -2.5 to  $V_{CC} + 1.0V$   
 Off-state output voltage applied . -2.5 to  $V_{CC} + 1.0V$   
 Storage temperature ..... -65 to 150°C

1. Stresses above those listed under the "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress only ratings and functional operation of the device at these or at any other conditions above those indicated in the operational sections of this specification is not implied (while programming, follow the programming specifications).

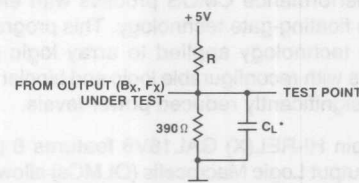
**PIN CONFIGURATION**

**OPERATING RANGE**

SYMBOL	PARAMETER	MILITARY			UNIT
		MIN.	TYP.	MAX.	
$V_{CC}$	Supply voltage	4.5	5.0	5.5	V
$T_A$	Ambient Temperature	-55			°C
$T_C$	Case Temperature			125	°C

**SWITCHING TEST CONDITIONS**

Input Pulse Levels	GND to 3.0V
Input Rise and Fall Times	5ns 10% - 90%
Input Timing Reference Levels	1.5V
Output Timing Reference Levels	1.5V
Output Load	See Figure

3-state levels are measured 0.5V from steady-state active level.

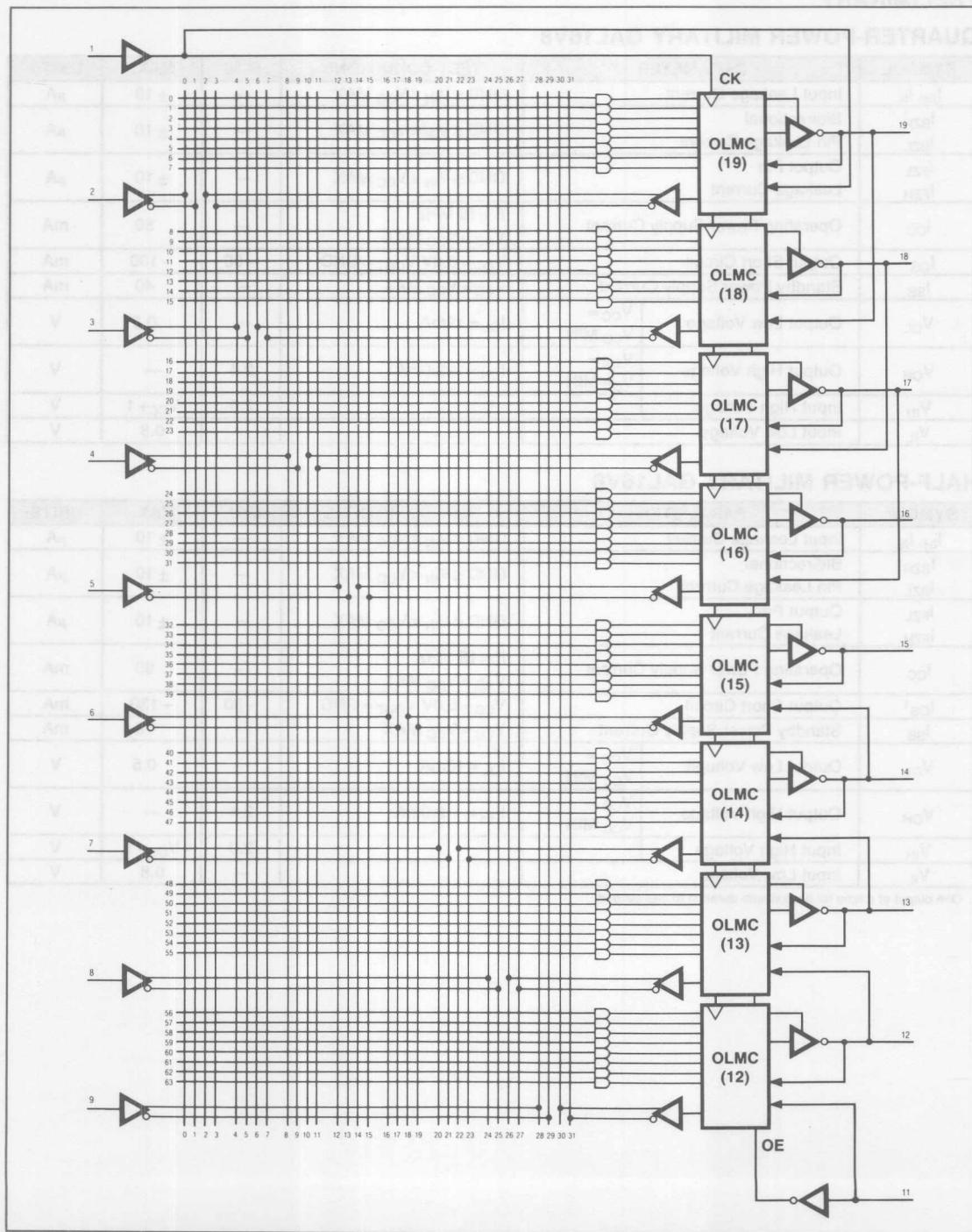


\* $C_L$  INCLUDES JIG AND PROBE TOTAL CAPACITANCE

**CAPACITANCE ( $T_A = 25^\circ C$ ,  $f = 1.0$  MHz)**

SYMBOL	PARAMETER	MAXIMUM	UNITS	TEST CONDITIONS
$C_I$	Input Capacitance	12	pF	$V_{CC} = 5.0V$ , $V_I = 2.0V$
$C_F$	Output Capacitance	15	pF	$V_{CC} = 5.0V$ , $V_F = 2.0V$
$C_B$	Bidirectional Pin Cap	15	pF	$V_{CC} = 5.0V$ , $V_B = 2.0V$

# GAL16V8 LOGIC DIAGRAM



## ELECTRICAL CHARACTERISTICS OVER OPERATING CONDITIONS

## PRELIMINARY

## QUARTER-POWER MILITARY GAL16V8

SYMBOL	PARAMETER	TEST CONDITIONS	MIN.	MAX.	UNITS
$I_{IH}, I_{IL}$	Input Leakage Current	$GND \leq V_{IN} \leq V_{CC} \text{ MAX}$	—	$\pm 10$	$\mu A$
$I_{BZH}, I_{BZL}$	Bidirectional Pin Leakage Current	$GND \leq V_{IN} < V_{CC} \text{ MAX}$	—	$\pm 10$	$\mu A$
$I_{FZL}, I_{FZH}$	Output Pin Leakage Current	$GND \leq V_{IN} \leq V_{CC} \text{ MAX}$	—	$\pm 10$	$\mu A$
$I_{CC}$	Operating Power Supply Current	$F = 15 \text{ MHz}$ $V_{CC} = V_{CC} \text{ MAX}$	—	50	mA
$I_{OS}^1$	Output Short Circuit	$V_{CC} = 5.0V, V_{OUT} = GND$	-30	-130	mA
$I_{SB}$	Standby Power Supply Current	$V_{CC} = V_{CC} \text{ MAX}$	—	40	mA
$V_{OL}$	Output Low Voltage	$V_{CC} = V_{CC} \text{ MIN}$ $I_{OL} = 12mA$	—	0.5	V
$V_{OH}$	Output High Voltage	$V_{CC} = V_{CC} \text{ MIN}$ $I_{OH} = -2.0mA$	2.4	—	V
$V_{IH}$	Input High Voltage		2.0	$V_{CC} + 1$	V
$V_{IL}$	Input Low Voltage		—	0.8	V

## HALF-POWER MILITARY GAL16V8

SYMBOL	PARAMETER	TEST CONDITIONS	MIN.	MAX.	UNITS
$I_{IH}, I_{IL}$	Input Leakage Current	$GND \leq V_{IN} \leq V_{CC} \text{ MAX}$	—	$\pm 10$	$\mu A$
$I_{BZH}, I_{BZL}$	Bidirectional Pin Leakage Current	$GND \leq V_{IN} < V_{CC} \text{ MAX}$	—	$\pm 10$	$\mu A$
$I_{FZL}, I_{FZH}$	Output Pin Leakage Current	$GND \leq V_{IN} \leq V_{CC} \text{ MAX}$	—	$\pm 10$	$\mu A$
$I_{CC}$	Operating Power Supply Current	$F = 15 \text{ MHz}$ $V_{CC} = V_{CC} \text{ MAX}$	—	90	mA
$I_{OS}^1$	Output Short Circuit	$V_{CC} = 5.0V, V_{OUT} = GND$	-30	-130	mA
$I_{SB}$	Standby Power Supply Current	$V_{CC} = V_{CC} \text{ MAX}$	—	70	mA
$V_{OL}$	Output Low Voltage	$V_{CC} = V_{CC} \text{ MIN}$ $I_{OL} = 12mA$	—	0.5	V
$V_{OH}$	Output High Voltage	$V_{CC} = V_{CC} \text{ MIN}$ $I_{OH} = -2.0mA$	2.4	—	V
$V_{IH}$	Input High Voltage		2.0	$V_{CC} + 1$	V
$V_{IL}$	Input Low Voltage		—	0.8	V

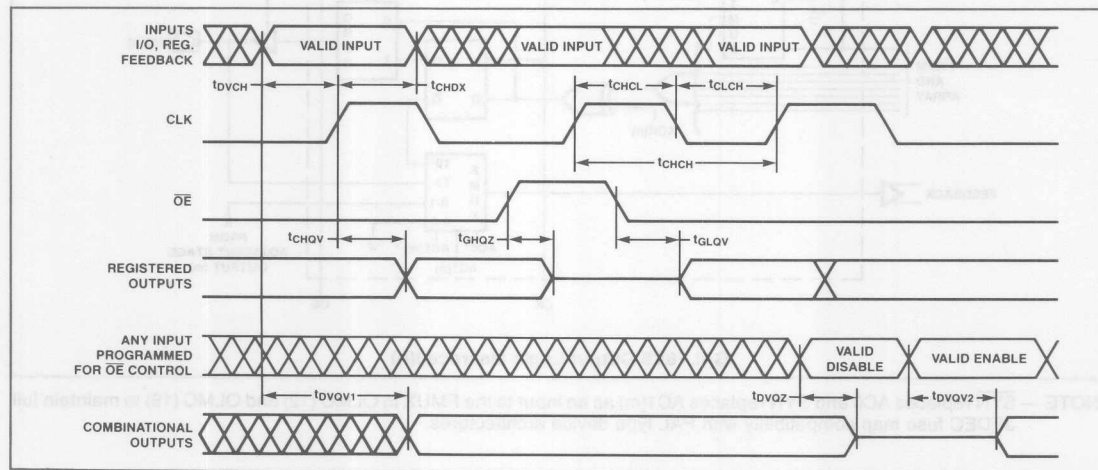
1 One output at a time for a maximum duration of one second.

## SWITCHING CHARACTERISTICS OVER OPERATING CONDITIONS

SYMBOL	PARAMETER	MILITARY						UNITS	TEST CONDITIONS <sup>1</sup>	
		GAL16V8-20		GAL16V8-30		GAL16V8-40			R(Ω)	C <sub>L</sub> (pF)
		MIN.	MAX.	MIN.	MAX.	MIN.	MAX.			
T <sub>DVQV1</sub>	Delay from Input to Active Output	—	20	—	30	—	40	ns	200	50
T <sub>DVQV2</sub>	Product Term Enable Access Time to Active Output	—	20	—	30	—	40	ns	Active High R = ∞ Active Low = 200	50
T <sub>DVQZ</sub> <sup>2</sup>	Product Term Disable to Outputs Off	—	20	—	30	—	40	ns	From V <sub>OH</sub> R = ∞ From V <sub>OL</sub> R = 200	5
T <sub>GHQZ</sub> <sup>2</sup>	G (OE) Output Enable High to Outputs Off	—	18	—	25	—	25	ns	From V <sub>OH</sub> R = ∞ From V <sub>OL</sub> R = 200	5
T <sub>GLQV</sub>	G (OE) Output Enable Access Time	—	18	—	25	—	25	ns	Active High R = ∞ Active Low R = 200	50
T <sub>CHQV</sub>	Clock High to Output Valid Access Time	—	15	—	20	—	25	ns	200	50
T <sub>DVCH</sub>	Input or Feedback Data Setup Time	15	—	25	—	35	—	ns	—	—
T <sub>CHDX</sub>	Input or Feedback Data Hold Time	0	—	0	—	0	—	ns	—	—
T <sub>CHCH</sub>	Clock Period (T <sub>DVCH</sub> + T <sub>CHQV</sub> )	30	—	45	—	60	—	ns		
T <sub>CHCL</sub>	Clock Width High	12	—	15	—	20	—	ns	—	—
T <sub>CLCH</sub>	Clock Width Low	12	—	15	—	20	—	ns		
f <sub>MAX</sub>	Maximum Frequency	SYNCH.	—	33.3	—	22.2	—	16.6	200	50
		ASYNCH.	—	50.0	—	33.3	—	25.0		

<sup>1</sup> Refer also to Switching Test Conditions. <sup>2</sup> 3-State levels are measured from steady-state active levels.

## SWITCHING WAVEFORMS





## OUTPUT LOGIC MACROCELL (OLMC)

The following discussion pertains to configuring the output logic macrocell. It should be noted that actual implementation is accomplished by development software/hardware and is completely transparent to the user.

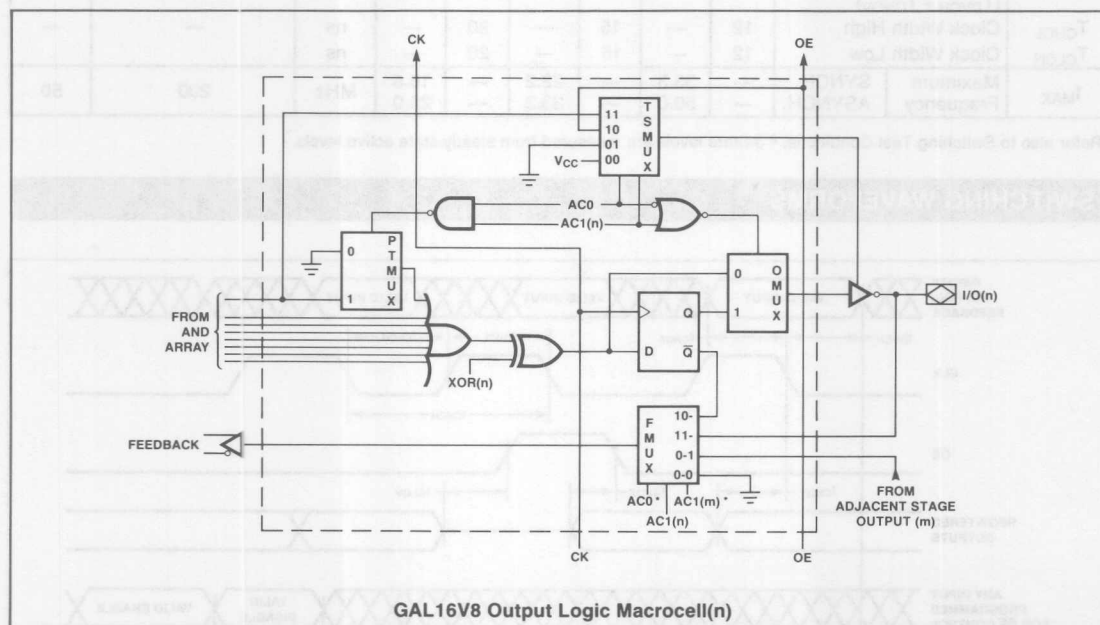
The outputs of the AND array are fed into an OLMC, where each output can be individually set to active high or active low, with either combinational (asynchronous) or registered (synchronous) configurations. A common output enable can be connected to all outputs, or separate inputs or product terms can be used to provide individual output enable controls. The output logic macrocell provides the designer with maximal output flexibility in matching signal requirements, thus providing more functions than possible with existing 20-pin PAL devices.

The various configurations of the output logic macrocell are controlled by programming certain cells (SYN, AC0, AC1(n), and the XOR(n) polarity bits) within the 82-bit architecture control word. The SYN bit determines

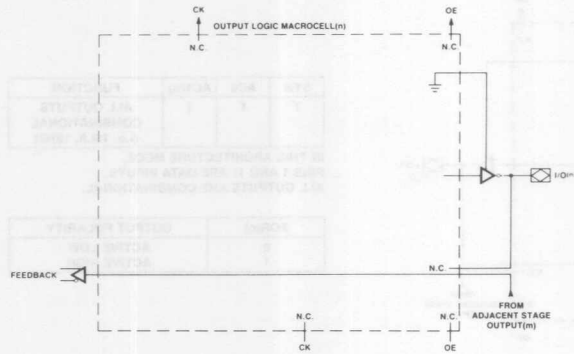
whether or not a device will have registered output capability or will have purely combinational outputs. It also replaces the AC0 bit in the two outermost macrocells, OLMC (12) and OLMC (19). When first setting up the device architecture, this is the first bit to choose.

Architecture control bit AC0 and the eight AC1(n) bits direct the outputs to be wired always on, always off (as an input), have a common OE term (pin 11), or to be three-state controlled separately from a product term. The architecture control bits also determine the source of the array feedback term through the FMUX, and select either combinational or registered outputs.

The five valid macrocell configurations are shown in each of the macrocell equivalent diagrams. In all cases, the eight XOR(n) bits individually determine each output's polarity. The truth table associated with each diagram shows the bit values of the SYN, AC0, and AC1(n) that set the macrocell to the configuration shown.



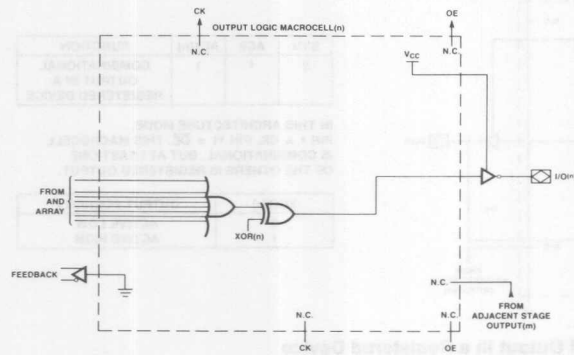
\* NOTE — SYN replaces AC0 and SYN replaces AC1(m) as an input to the FMUX in OLMC (12) and OLMC (19) to maintain full JEDEC fuse map compatibility with PAL type device architectures.



SYN	AC0	AC1(n)	FUNCTION
1	0	1	INPUT MODE (i.e. 12L6, 14P4)

IN THIS ARCHITECTURE MODE,  
PINS 1 AND 11 ARE DATA INPUTS.  
THE OUTPUT BUFFER IS DISABLED.

Dedicated Input Mode

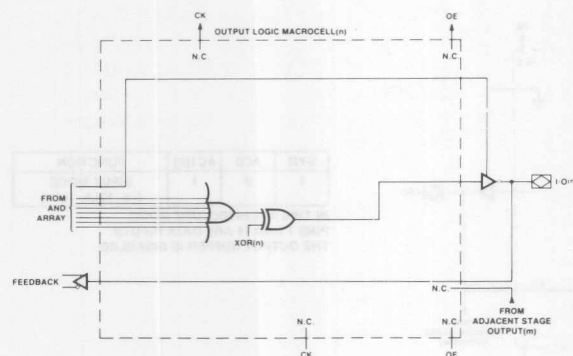


SYN	AC0	AC1(n)	FUNCTION
1	0	0	ALL OUTPUTS COMBINATIONAL (i.e. 10L8, 12H6)

IN THIS ARCHITECTURE MODE,  
PINS 1 AND 11 ARE DATA INPUTS.  
ALL OUTPUTS ARE COMBINATIONAL AND  
ALWAYS ACTIVE.

XOR(n)	OUTPUT POLARITY
0	ACTIVE LOW
1	ACTIVE HIGH

Dedicated Combinational Output

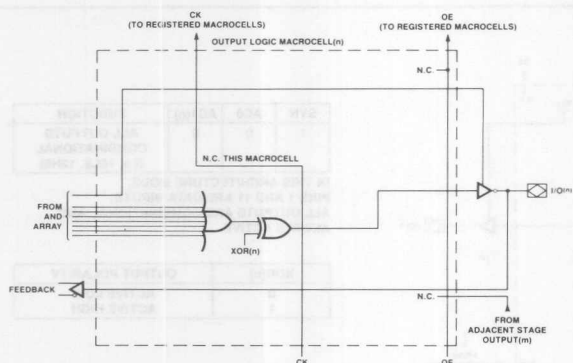


Combinational Output

SYN	AC0	AC1(n)	FUNCTION
1	1	1	ALL OUTPUTS COMBINATIONAL (i.e. 16L8, 16H8)

IN THIS ARCHITECTURE MODE, PINS 1 AND 11 ARE DATA INPUTS. ALL OUTPUTS ARE COMBINATIONAL.

XOR(n)	OUTPUT POLARITY
0	ACTIVE LOW
1	ACTIVE HIGH

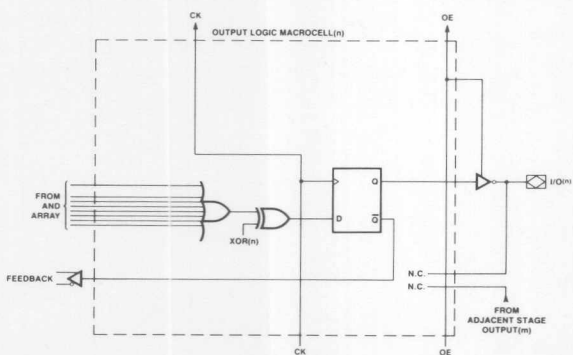


Combinational Output in a Registered Device

SYN	AC0	AC1(n)	FUNCTION
0	1	1	COMBINATIONAL OUTPUT IN A REGISTERED DEVICE

IN THIS ARCHITECTURE MODE, PIN 1 = CK, PIN 11 = OE. THIS MACROCELL IS COMBINATIONAL, BUT AT LEAST ONE OF THE OTHERS IS REGISTERED OUTPUT.

XOR(n)	OUTPUT POLARITY
0	ACTIVE LOW
1	ACTIVE HIGH



Registered Active High or Low Output

SYN	AC0	AC1(n)	FUNCTION
0	1	0	OUTPUT REGISTERED (i.e. 16R8)

IN THIS ARCHITECTURE MODE, PIN 1 = CK, PIN 11 = OE.

XOR(n)	OUTPUT POLARITY
0	ACTIVE LOW
1	ACTIVE HIGH

## ROW ADDRESS MAP DESCRIPTION

Figure 1 shows a block diagram of the row address map. There are a total of 36 unique row addresses available to the user when programming the GAL16V8 devices. Row addresses 0–31 each contain 64 bits of input term data. This is the user array where the custom logic pattern is programmed. Row 32 is the electronic signature word. It has 64 bits available for any user-defined purpose. Row 33–59 are reserved by the manufacturer and are not available to users.

Row 60 contains the architecture and output polarity information. The 82 bits within this word are programmed to configure the device for a specific application. Row 61 contains a one bit security cell that when programmed prevents further programming verification of the array. Row 63 is the row that is addressed to perform a bulk erase of the device, resetting it back to a virgin state. Each of these functions is described in the following sections.

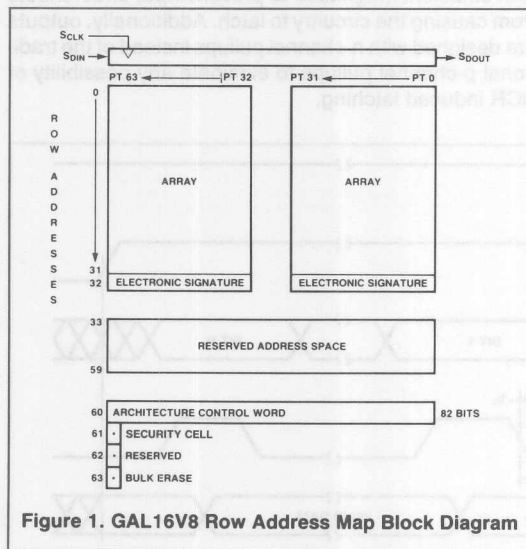


Figure 1. GAL16V8 Row Address Map Block Diagram

## ELECTRONIC SIGNATURE WORD

An electronic signature word is provided with every GAL16V8 device. It resides at Row address 32 and contains 64 bits of reprogrammable memory that can contain user-defined data. Some uses include user ID codes, revision numbers, or inventory control. This signature data is always available to the user independent of the state of the security cell.

## ARCHITECTURE CONTROL WORD

All of the various configurations of the GAL16V8 devices are controlled by programming cells within the 82-bit architecture control word that resides at row 60. The location of specific bits within the architecture control word is shown in the control word diagram in Figure 2. The function of the SYN, AC0 and AC1(n) bits have been explained in the output logic macrocell description. The eight polarity bits determine each output's polarity individually by selectively correct logic. The numbers below the XOR(n) and AC1(n) bits in the architecture control word diagram shows the output device pin number that the polarity bits control.

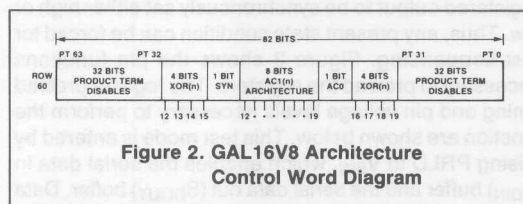


Figure 2. GAL16V8 Architecture Control Word Diagram

## SECURITY CELL

Row address 61 contains the security cell (one bit). The security cell is provided on all GAL16V8 devices as a deterrent to unauthorized copying of the array configuration patterns. Once programmed, the circuitry enabling array access is disabled, preventing further programming or verification of the array (rows 0–31). The cell can be erased only in conjunction with the array during a bulk erase cycle, so the original configuration can never be examined once this cell is programmed. Signature data is **always** available to the user.

## BULK ERASE MODE

By addressing row 63 during a programming cycle, a clear function performs a bulk erase of the array and the architecture word. In addition, the electronic signature word and the security cell are erased. This mode resets a previously configured device back to its virgin state.

## OUTPUT REGISTER PRELOAD

When testing state machine designs, all possible states and state transitions must be verified in the design, not just those required in the normal machine operations. This is because in system operation, certain events occur that may throw the logic into an illegal state (power-up, line voltage glitches, brown-outs, etc.). To test a design for proper treatment of these conditions, a way must be provided to break the feedback paths, and force any desired (ie. illegal) state into the registers. Then the machine can be sequenced and the outputs tested for correct next state conditions.

The GAL16V8 device includes circuitry that allows each registered output to be synchronously set either high or low. Thus, any present state condition can be forced for test sequencing. Figure 3 shows the pin functions necessary to preload the registers. The register preload timing and pin voltage levels necessary to perform the function are shown below. This test mode is entered by raising PRLD to  $V_{IES}$ , which enables the serial data in ( $S_{DIN}$ ) buffer and the serial data out ( $S_{DOUT}$ ) buffer. Data is then serially shifted into the registers on each rising edge of the clock, DCLK. Only the macrocells with registered output configurations are loaded. If only 3

outputs have registers, then only 3 bits need be shifted in. The registers are loaded from the bottom up, as shown in Figure 3.

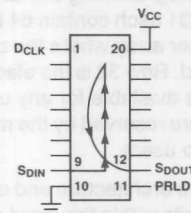


Figure 3. Output Register Preload Pinout

## LATCH-UP PROTECTION

GAL devices are designed with an on-board charge pump to negatively bias the substrate. The negative bias is of sufficient magnitude to prevent input undershoots from causing the circuitry to latch. Additionally, outputs are designed with n-channel pullups instead of the traditional p-channel pullups to eliminate any possibility of SCR induced latching.

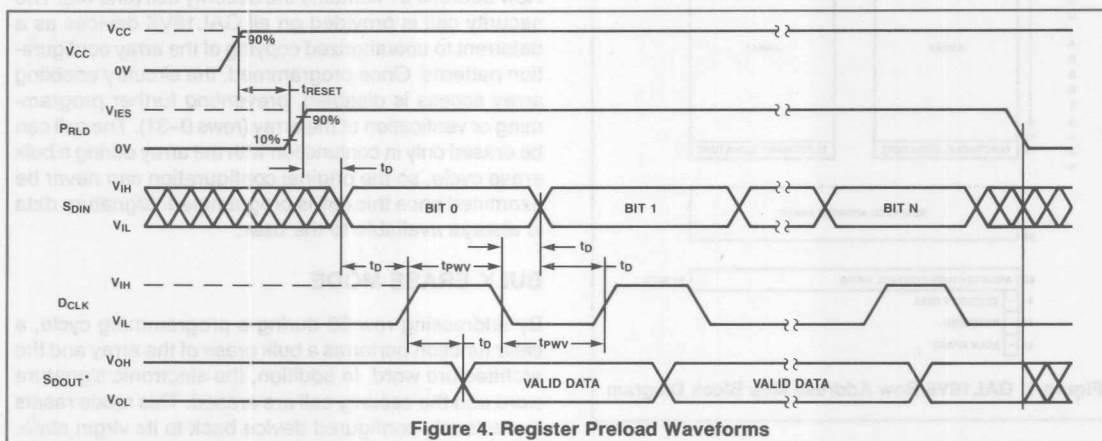
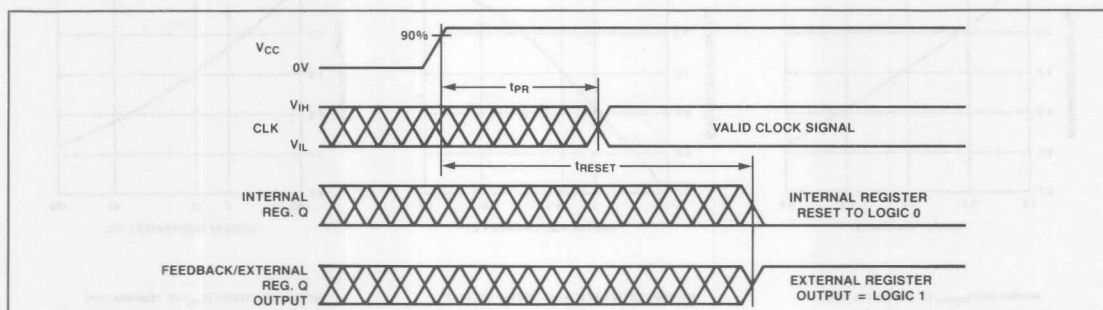


Figure 4. Register Preload Waveforms

\* NOTE — The  $S_{DOUT}$  output buffer is an open drain output during preload. This pin should be terminated to  $V_{CC}$  with a 10K resistor.



## POWER-UP RESET



Circuitry within the GAL16V8 provides a reset signal to all registers during power-up. All internal registers will have their Q outputs set low after a specified time ( $t_{RESET}$ ). As a result, the state on the registered output pins (if they are enabled through  $\overline{OE}$ ) will always be high on power-up, regardless of the programmed polarity of the output pins. This feature can greatly simplify state machine design by providing a known state on power-up.

The timing diagram for power-up is shown above. Because of asynchronous nature of system power-up, some conditions must be met to guarantee a valid power-up reset of the GAL16V8. First, the  $V_{CC}$  rise must be monotonic. Second, the clock input must become a proper TTL level within the specified time ( $t_{PR}$ ). The registers will reset within a maximum of  $t_{RESET}$  time. As in normal system operation, avoid clocking the device until all input and feedback path setup times have been met.

2

## FIELD SUPPORT TOOLS

## PROGRAMMER/DEVELOPMENT SYSTEMS

VENDOR	SYSTEM	REVISION
DATA I/O	Model 29B	V04
	Logic Pak	
	P/T Adapter 303A-009	V03
STAG	Model 60	**
	PPZ/ZM2200	20/11
	ZL30 & ZL32	V30.41
	ZL30A	V30A.03
VARIX	Omni-Programmer	**
INLAB	Model 28	**
VALLEY DATA SCIENCES	Model 160	**

\*\* This version being qualified.

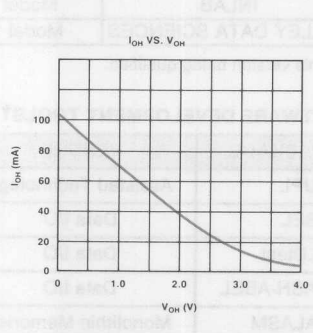
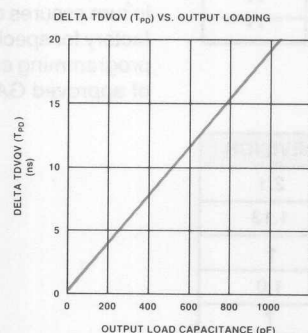
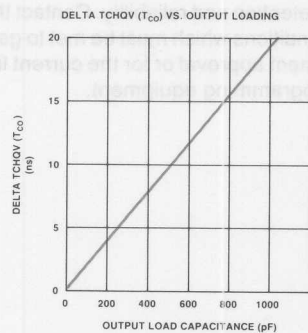
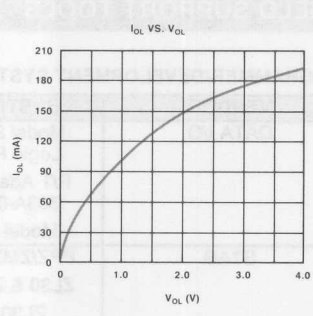
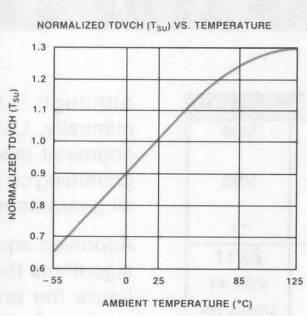
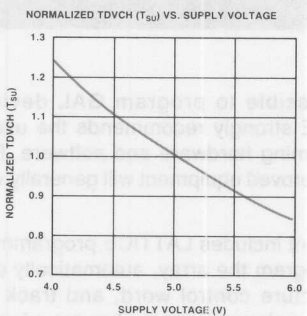
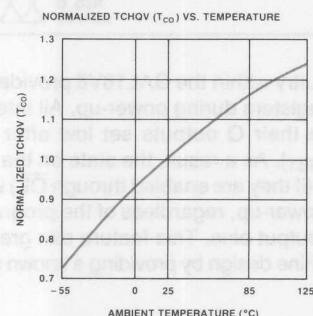
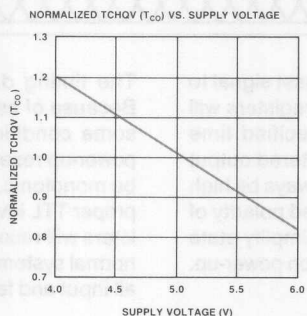
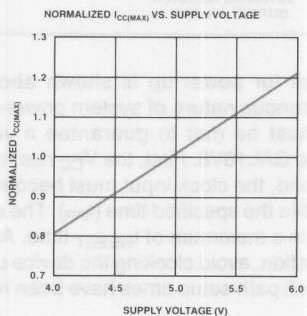
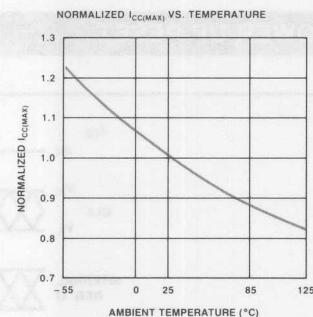
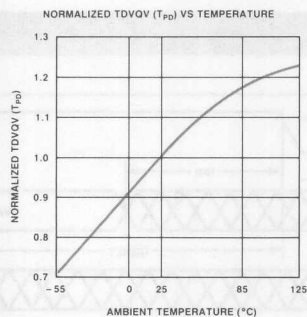
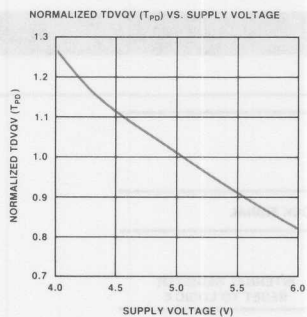
Although it is possible to program GAL devices manually, LATTICE strongly recommends the use of approved programming hardware and software. Programming on unapproved equipment will generally void all guarantees.

Approved equipment includes LATTICE programming algorithms that program the array, automatically configure the architecture control word, and track the number of program cycles each device has experienced (this information is stored within each GAL device). This in turn assures data retention and reliability. Contact the factory for specific conditions which must be met to gain programming equipment approval or for the current list of approved GAL programming equipment.

## SOFTWARE DEVELOPMENT TOOLST

PACKAGE	VENDOR	REVISION
CUPL	Assisted Technology	2.1
ABEL	Data I/O	1.13
PLDtest	Data I/O	†
DASH-ABEL	Data I/O	1.0
PALASM	Monolithic Memories	†

† When emulating PAL devices any revision of the software can be used to create the PAL JEDEC file. The programming hardware will automatically configure the GAL architecture.



All LATTICE Semiconductor products begin with exacting design criteria established for the devices' ultimate use in high-reliability programs. All products are manufactured, inspected, and tested in compliance with LATTICE's Quality Assurance program, which meets or exceeds all requirements as outlined in MIL-M 38510F Appendix A, as well as all inspection system requirements in MIL-I-45208A. LATTICE enforces all such requirements on any and all subcontractors involved in the manufacture of its product. LATTICE is solely responsible for securing and proving the documentation and control of its Quality Assurance program. All HI-REL(X) grade products undergo demanding screening procedures according to the Methods and

Requirements described below, which can be found in MIL-STD-883C, Method 5004. A high-reliability assembly flow is employed with 100% internal-visual, stabilization-bake temperature-cycling, and constant acceleration screens. The HI-REL(X) grade processing also includes as standard 100% 160-hour burn-in at an ambient temperature of +125°C per Method 1015, followed by 100% temperature testing of all DC and AC parameters over the full -55°C to +125°C temperature range. For special customer specifications or quality requirements — such as SEM analysis, X-ray, or other screening flows to meet specific needs — please contact your local sales office or LATTICE Semiconductor directly at 1-800-FASTGAL.

## SCREENING FLOW

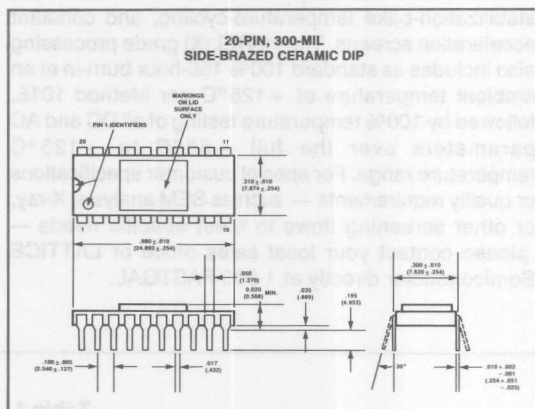
Table 1.

Screen	Method	Requirement
Wafer Lot Acceptance	Per LATTICE Specification	3 Wafers/Lot
Internal Visual	2010 Cond. B	100%
Stabilization Bake	1008 24 HR Cond. C	100%
Temp Cycling	1010 Cond. C	100%
Constant Acceleration	2001 Cond. E	100%
Visual Inspection	5004	100%
Hermeticity	1014	100%
Fine	Cond. A1	
Gross	Cond. C	
Pre-Burn-In Electrical	Applicable Device Specification $T_A = 25^\circ\text{C}$	100%
Burn-In	1015, 160 HRS, 125°C	100%
Post Burn-In Electrical	Applicable Device Specification $T_A = 25^\circ\text{C}$	100%
Percent Defective Allowable	5004	5%
Final Electrical Test	Applicable Device Specification $T_A = 125^\circ\text{C}$	100%
Final Electrical Test	Applicable Device Specification $T_A = -55^\circ\text{C}$	100%
OCI Sample Selection	MIL-M-38510F Section 4.5	Sample
External Visual	2009	100%

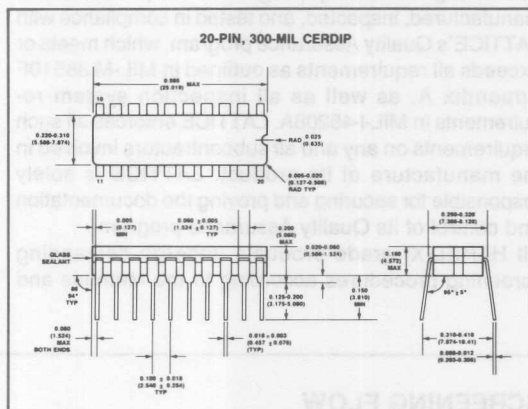
2

## PACKAGE INFORMATION

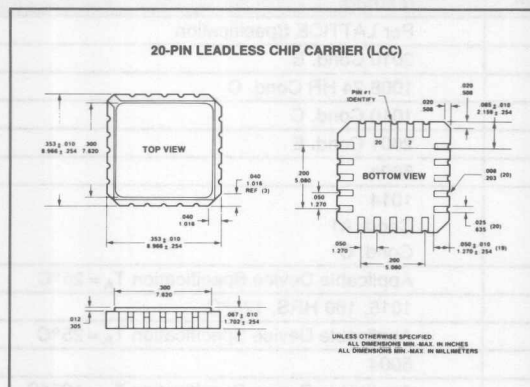
20-PIN, 300-MIL  
SIDE-BRAZED CERAMIC DIP



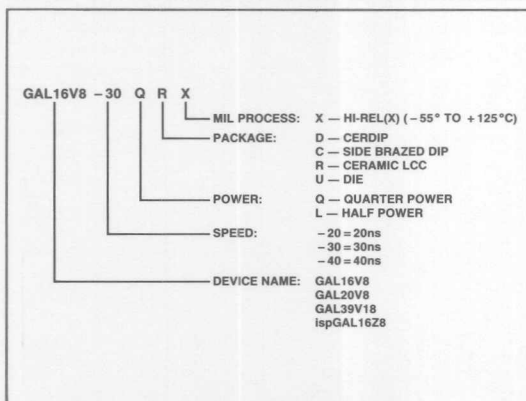
20-PIN, 300-MIL CERDIP



**20-PIN LEADLESS CHIP CARRIER (LCC)**



## ORDERING INFORMATION



The specifications and information contained herein are subject to change without notice.

## SPEED/POWER CROSS-REFERENCE GUIDE

SPEED	POWER	GAL DEVICE	BIPOLAR PAL DEVICE
15ns	45mA	— 15Q	—
15ns	90mA	— 15L	—
15ns	180mA	use — 15L	B
* 20ns	50mA	— 20Q	—
* 20ns	90mA	— 20L	—
* 20ns	210mA	use — 20L	B MIL
25ns	45mA	— 25Q	—
25ns	90mA	— 25L	B-2
25ns	180mA	use — 25L	A
* 30ns	50mA	— 30Q	—
* 30ns	90mA	— 30L	B-2 MIL
* 30ns	210mA	use — 30L	A MIL
35ns	45mA	— 35Q	B-4
35ns	90mA	— 35L	A-2
35ns	180mA	use — 35L	STD
* 40ns	50mA	— 40Q	B-4 MIL
* 40ns	90mA	— 40L	A-2 MIL
* 40ns	210mA	use — 40L	STD MIL

\* MILITARY TEMPERATURE RANGE

## PRELIMINARY

## FEATURES

- SPECIFICATIONS GUARANTEED OVER FULL MILITARY TEMPERATURE RANGE ( - 55°C TO + 125°C)
- ELECTRICALLY ERASABLE CELL TECHNOLOGY
  - Reconfigurable Logic
  - Reprogrammable Cells
  - Guaranteed 100% Yields
- HIGH PERFORMANCE E<sup>2</sup>C MOS TECHNOLOGY
  - Low Power: 50 mA Max Active  
40 mA Max Standby
  - High Speed: 20 ns Access Max  
30 ns Access Max
- 160-HR BURN IN
- HI-REL ASSEMBLY FLOW
- HIGH-SPEED PROGRAMMING ALGORITHM
- SECURITY CELL PREVENTS COPYING LOGIC
- DATA RETENTION EXCEEDS 20 YEARS

## DESCRIPTION

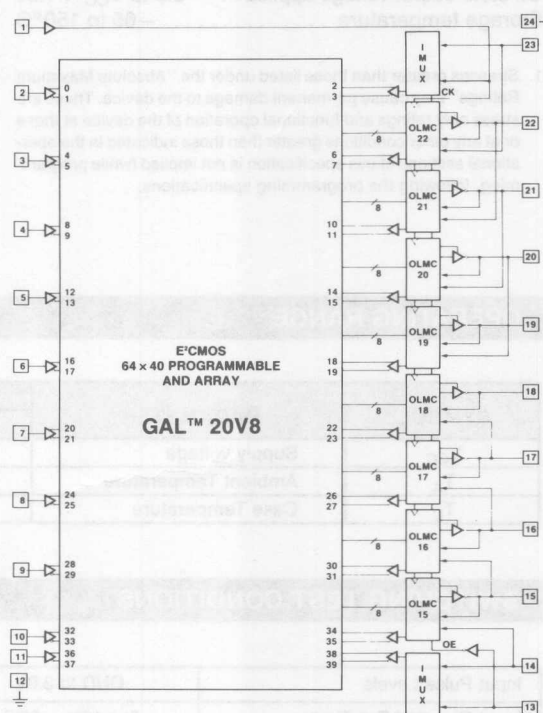
Lattice Semiconductor HI-REL(X) GAL20V8 combines a high-performance CMOS process with electrically erasable floating gate technology. This programmable memory technology applied to array logic provides designers with reconfigurable logic and bipolar performance at significantly reduced power levels.

The 24-pin HI-REL(X) GAL20V8 features 8 programmable Output Logic Macrocells (OLMC) allowing each output to be configured by the user. Additionally, the GAL20V8 is capable of emulating, in a functional/fuse map compatible device, all common 24-pin PAL device architectures.

The HI-REL(X) GAL20V8 is fully screened to a demanding assembly, burn-in, and test flow, as shown in Table 1. Access times as fast as 20ns are available — guaranteed over the full military temperature range of -55°C to +125°C.

Unique test circuitry and reprogrammable cells allow complete AC, DC, and functionality testing during manufacture. Therefore, Lattice guarantees 100% field programmability and functionality of the GAL devices. In addition, electronic signature is available to provide positive device ID. A security circuit is built-in, providing proprietary designs with copy protection. The HI-REL(X) GAL20V8 is available in 24-pin Cerdip and side-brazed DIP packages or a space saving 28-lead square ceramic leadless chip carrier (LCC).

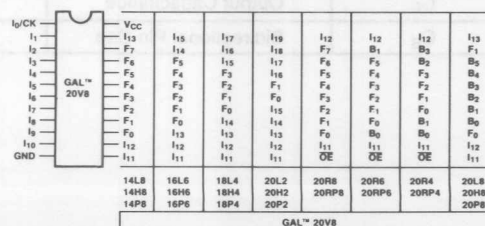
## FUNCTIONAL BLOCK DIAGRAM



## PIN NAMES

I <sub>0</sub> -I <sub>19</sub>	INPUT	$\overline{OE}$	OUTPUT ENABLE
CK	CLOCK INPUT	V <sub>CC</sub>	POWER (+5V)
B <sub>0</sub> -B <sub>5</sub>	BIDIRECTIONAL	GND	GROUND
F <sub>0</sub> -F <sub>7</sub>	OUTPUT		

## GAL20V8 EMULATING PAL DEVICES



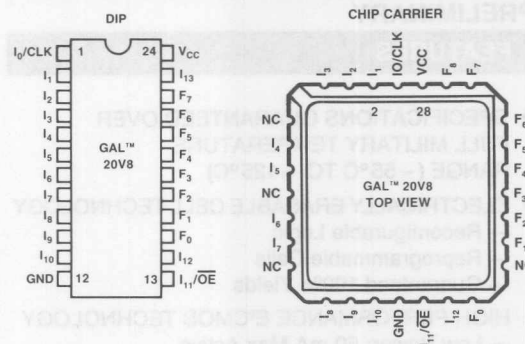


## ABSOLUTE MAXIMUM RATINGS <sup>1</sup>

Supply voltage  $V_{CC}$  . . . . . - .5 to +7V  
Input voltage applied . . . . . - 2.5 to  $V_{CC} + 1.0V$   
Off-state output voltage applied . . . - 2.5 to  $V_{CC} + 1.0V$   
Storage temperature . . . . . - 65 to 150°C

1. Stresses greater than those listed under the "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress only ratings and functional operation of the device at these or at any other conditions greater than those indicated in the operational sections of this specification is not implied (while programming, following the programming specifications).

## PIN CONFIGURATION



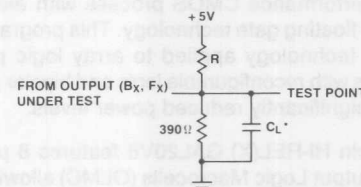
## OPERATING RANGE

SYMBOL	PARAMETER	MILITARY			UNIT
		MIN.	TYP.	MAX.	
$V_{CC}$	Supply voltage	4.5	5.0	5.5	V
$T_A$	Ambient Temperature	-55			°C
$T_C$	Case Temperature			125	°C

## SWITCHING TEST CONDITIONS

Input Pulse Levels	GND to 3.0V
Input Rise and Fall Times	5ns 10% - 90%
Input Timing Reference Levels	1.5V
Output Timing Reference Levels	1.5V
Output Load	See Figure

3-state levels are measured 0.5V from steady-state active level.

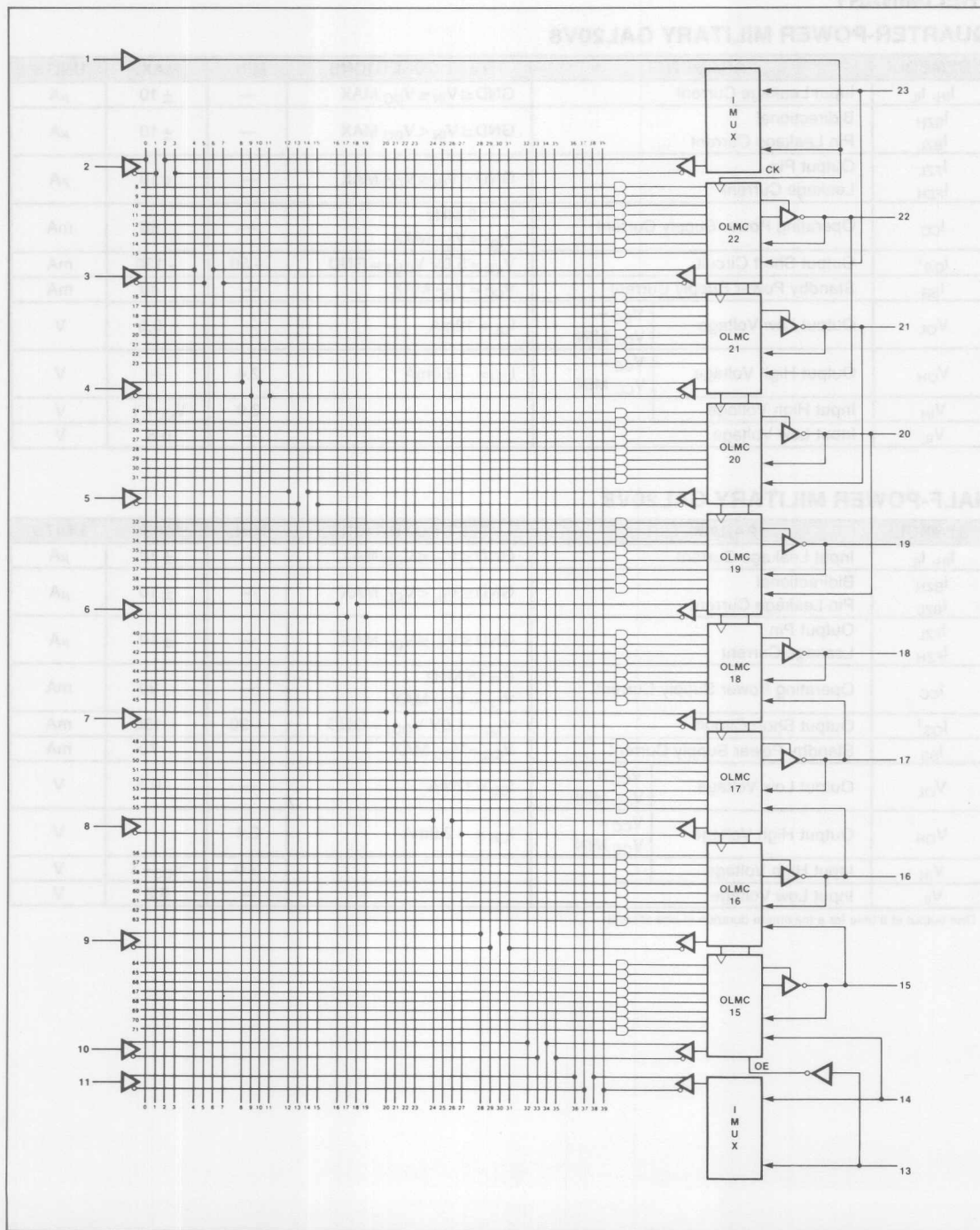


\* $C_L$  INCLUDES JIG AND PROBE TOTAL CAPACITANCE

## CAPACITANCE ( $T_A = 25^\circ\text{C}$ , $f = 1.0\text{ MHz}$ )

SYMBOL	PARAMETER	MAXIMUM	UNITS	TEST CONDITIONS
$C_I$	Input Capacitance	12	pF	$V_{CC} = 5.0V$ , $V_I = 2.0V$
$C_F$	Output Capacitance	15	pF	$V_{CC} = 5.0V$ , $V_F = 2.0V$
$C_B$	Bidirectional Pin Cap	15	pF	$V_{CC} = 5.0V$ , $V_B = 2.0V$

# GAL 20V8 LOGIC DIAGRAM



## ELECTRICAL CHARACTERISTICS OVER OPERATING CONDITIONS

## PRELIMINARY

## QUARTER-POWER MILITARY GAL20V8

SYMBOL	PARAMETER	TEST CONDITIONS	MIN.	MAX.	UNITS
$I_{IH}, I_{IL}$	Input Leakage Current	$GND \leq V_{IN} \leq V_{CC} \text{ MAX}$	—	$\pm 10$	$\mu A$
$I_{BZH}, I_{BZL}$	Bidirectional Pin Leakage Current	$GND \leq V_{IN} < V_{CC} \text{ MAX}$	—	$\pm 10$	$\mu A$
$I_{FZL}, I_{FZH}$	Output Pin Leakage Current	$GND \leq V_{IN} \leq V_{CC} \text{ MAX}$	—	$\pm 10$	$\mu A$
$I_{CC}$	Operating Power Supply Current	$F = 15 \text{ MHz}$ $V_{CC} = V_{CC} \text{ MAX}$	—	50	mA
$I_{OS}^1$	Output Short Circuit	$V_{CC} = 5.0V, V_{OUT} = GND$	-30	-130	mA
$I_{SB}$	Standby Power Supply Current	$V_{CC} = V_{CC} \text{ MAX}$	—	40	mA
$V_{OL}$	Output Low Voltage	$V_{CC} = V_{CC} \text{ MIN}$ $I_{OL} = 12\text{mA}$	—	0.5	V
$V_{OH}$	Output High Voltage	$V_{CC} = V_{CC} \text{ MIN}$ $I_{OH} = -2.0\text{mA}$	2.4	—	V
$V_{IH}$	Input High Voltage		2.0	$V_{CC} + 1$	V
$V_{IL}$	Input Low Voltage		—	0.8	V

## HALF-POWER MILITARY GAL20V8

SYMBOL	PARAMETER	TEST CONDITIONS	MIN.	MAX.	UNITS
$I_{IH}, I_{IL}$	Input Leakage Current	$GND \leq V_{IN} \leq V_{CC} \text{ MAX}$	—	$\pm 10$	$\mu A$
$I_{BZH}, I_{BZL}$	Bidirectional Pin Leakage Current	$GND \leq V_{IN} < V_{CC} \text{ MAX}$	—	$\pm 10$	$\mu A$
$I_{FZL}, I_{FZH}$	Output Pin Leakage Current	$GND \leq V_{IN} \leq V_{CC} \text{ MAX}$	—	$\pm 10$	$\mu A$
$I_{CC}$	Operating Power Supply Current	$F = 15 \text{ MHz}$ $V_{CC} = V_{CC} \text{ MAX}$	—	90	mA
$I_{OS}^1$	Output Short Circuit	$V_{CC} = 5.0V, V_{OUT} = GND$	-30	-130	mA
$I_{SB}$	Standby Power Supply Current	$V_{CC} = V_{CC} \text{ MAX}$	—	70	mA
$V_{OL}$	Output Low Voltage	$V_{CC} = V_{CC} \text{ MIN}$ $I_{OL} = 12\text{mA}$	—	0.5	V
$V_{OH}$	Output High Voltage	$V_{CC} = V_{CC} \text{ MIN}$ $I_{OH} = -2.0\text{mA}$	2.4	—	V
$V_{IH}$	Input High Voltage		2.0	$V_{CC} + 1$	V
$V_{IL}$	Input Low Voltage		—	0.8	V

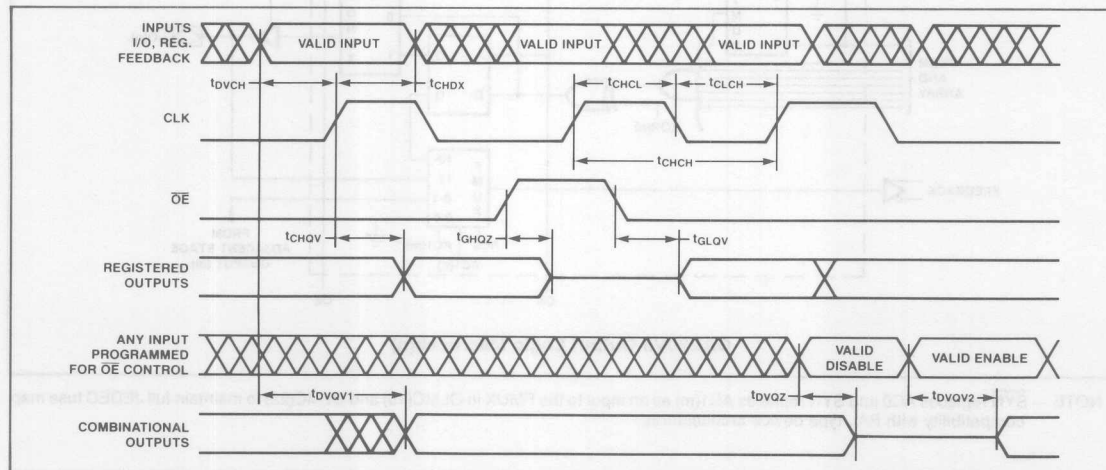
<sup>1</sup> One output at a time for a maximum duration of one second.

## SWITCHING CHARACTERISTICS OVER OPERATING CONDITIONS

SYMBOL	PARAMETER		MILITARY						UNITS	TEST CONDITIONS <sup>1</sup>	
			GAL20V8-20		GAL20V8-30		GAL20V8-40			R(Ω)	C <sub>L</sub> (pF)
			MIN.	MAX.	MIN.	MAX.	MIN.	MAX.			
T <sub>DVQV1</sub>	Delay from Input to Active Output		—	20	—	30	—	40	ns	200	50
T <sub>DVQV2</sub>	Product Term Enable Access Time to Active Output		—	20	—	30	—	40	ns	Active High R = ∞ Active Low = 200	50
T <sub>DVQZ</sub> <sup>2</sup>	Product Term Disable to Outputs Off		—	20	—	30	—	40	ns	From V <sub>OH</sub> R = ∞ From V <sub>OL</sub> R = 200	5
T <sub>GHQZ</sub> <sup>2</sup>	G (OE) Output Enable High to Outputs Off		—	18	—	25	—	25	ns	From V <sub>OH</sub> R = ∞ From V <sub>OL</sub> R = 200	5
T <sub>GLQV</sub>	G (OE) Output Enable Access Time		—	18	—	25	—	25	ns	Active High R = ∞ Active Low R = 200	50
T <sub>CHQV</sub>	Clock High to Output Valid Access Time		—	15	—	20	—	25	ns	200	50
T <sub>DVCH</sub>	Input or Feedback Data Setup Time		15	—	25	—	35	—	ns	—	—
T <sub>CHDX</sub>	Input or Feedback Data Hold Time		0	—	0	—	0	—	ns	—	—
T <sub>CHCH</sub>	Clock Period (T <sub>DVCH</sub> + T <sub>CHQV</sub> )		30	—	45	—	60	—	ns	—	—
T <sub>CHCL</sub>	Clock Width High		12	—	15	—	20	—	ns		
T <sub>CLCH</sub>	Clock Width Low		12	—	15	—	20	—	ns		
f <sub>MAX</sub>	Maximum Frequency	SYNCH. ASYNCH.	— —	33.3 50.0	— —	22.2 33.3	— —	16.6 25.0	MHz	200	50

<sup>1</sup>Refer also to Switching Test Conditions. <sup>2</sup>3-State levels are measured 0.5V from steady-state active level.

## SWITCHING WAVEFORMS



## OUTPUT LOGIC MACROCELL

The following discussion pertains to configuring the Output Logic Macrocell. It should be noted that actual implementation is accomplished by development software/hardware and is completely transparent to the user.

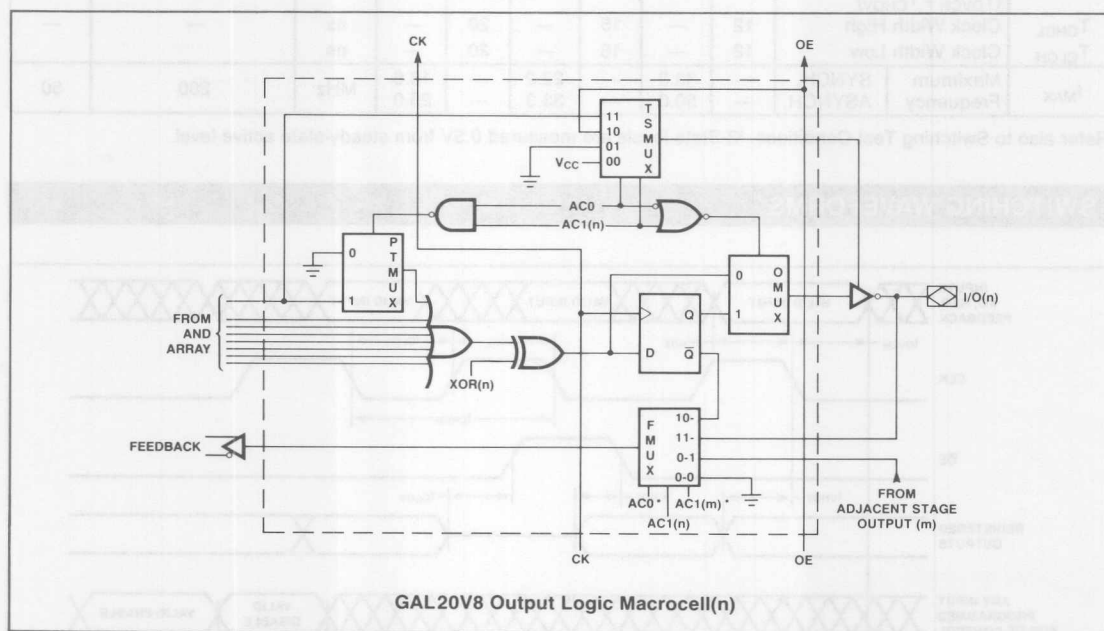
The outputs of the AND array are fed into an OLMC, where each output can be individually set to active high, or active low, with either combinational (asynchronous) or registered (synchronous) configurations. A common output enable can be connected to all outputs, or separate inputs or product terms can be used to provide individual output enable controls. The Output Logic Macrocell provides the designer with maximal output flexibility in matching signal requirements, thus providing more functions than possible with existing 24-pin PAL devices.

The various configurations of the Output Logic Macrocell are controlled by programming certain cells (SYN, AC0, AC1(n), and the XOR(n) polarity bits) within the 82-bit Architecture Control Word. The SYN bit determines

whether a device will have registered output capability or purely combinational outputs. It also replaces the AC0 bit in the two outermost macrocells, OLMC (15) and OLMC (22). When first setting up the device architecture, this is the first bit to choose.

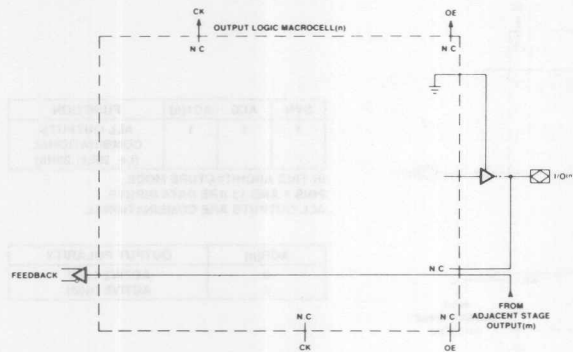
Architecture Control bit AC0 and the eight AC1(n) bits direct the outputs to be wired always on, always off (as an input), have a common OE term (pin 13), or to be three-state controlled separately from a product term. The Architecture Control bits also determine the source of the array feedback term through the FMUX, and select either combinational or registered outputs.

The five valid macrocell configurations are shown in each of the macrocell equivalent diagrams. In all cases, the eight XOR(n) bits individually determine each output's polarity. The truth table associated with each diagram shows the bit values of the SYN, AC0, and AC1(n) that set the macrocell to the configuration shown.



\* NOTE —  $\overline{\text{SYN}}$  replaces AC0 and SYN replaces AC1(m) as an input to the FMUX in OLMC(15) and OLMC(22) to maintain full JEDEC fuse map compatibility with PAL type device architectures.

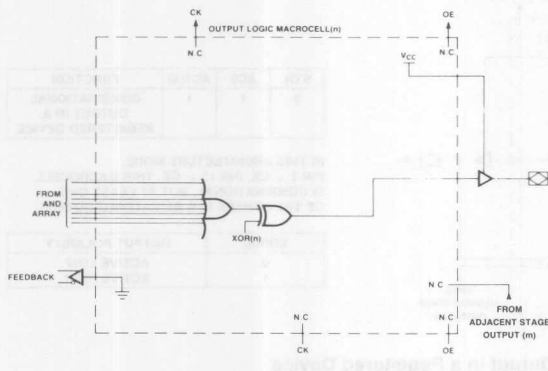




Dedicated Input Mode

SYN	AC0	AC1(n)	FUNCTION
1	0	1	INPUT MODE (i.e. 20L2, 18P4)

IN THIS ARCHITECTURE MODE,  
PINS 1 AND 13 ARE DATA INPUTS.  
THE OUTPUT BUFFER IS DISABLED.

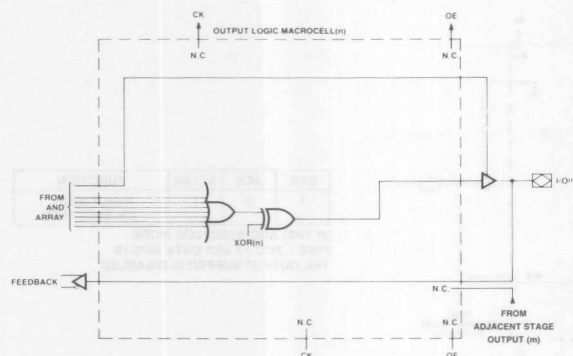


Dedicated Combinational Output

SYN	AC0	AC1(n)	FUNCTION
1	0	0	ALL OUTPUTS COMBINATIONAL (i.e. 16H6)

IN THIS ARCHITECTURE MODE,  
PINS 1 AND 13 ARE DATA INPUTS.  
ALL MACROCELLS CONFIGURED AS OUTPUTS  
ARE COMBINATIONAL AND ALWAYS ACTIVE.

XOR(n)	OUTPUT POLARITY
0	ACTIVE LOW
1	ACTIVE HIGH

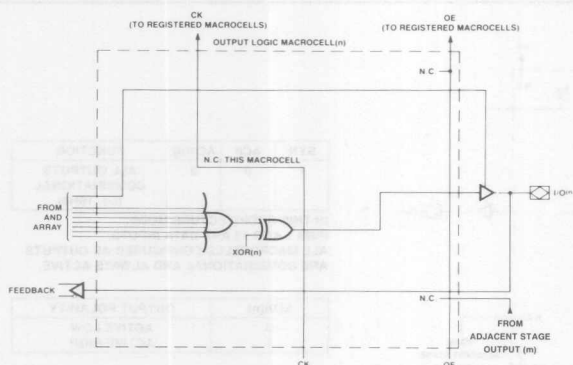


SYN	AC0	AC1(n)	FUNCTION
1	1	1	ALL OUTPUTS COMBINATIONAL (i.e. 20L8, 20H8)

IN THIS ARCHITECTURE MODE, PINS 1 AND 13 ARE DATA INPUTS. ALL OUTPUTS ARE COMBINATIONAL.

XOR(n)	OUTPUT POLARITY
0	ACTIVE LOW
1	ACTIVE HIGH

Combinational Output

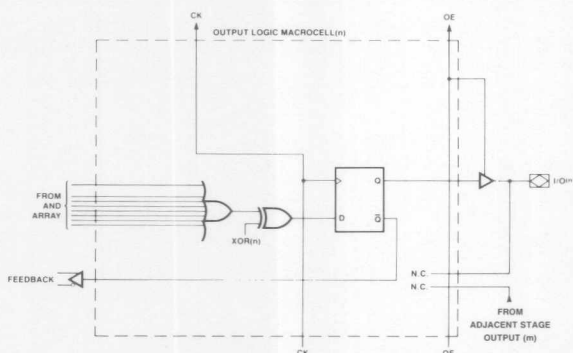


SYN	AC0	AC1(n)	FUNCTION
0	1	1	COMBINATIONAL OUTPUT IN A REGISTERED DEVICE

IN THIS ARCHITECTURE MODE, PIN 1 = CK, PIN 13 = OE. THIS MACROCELL IS COMBINATIONAL, BUT AT LEAST ONE OF THE OTHERS IS A REGISTERED OUTPUT.

XOR(n)	OUTPUT POLARITY
0	ACTIVE LOW
1	ACTIVE HIGH

Combinational Output in a Registered Device



SYN	AC0	AC1(n)	FUNCTION
0	1	0	OUTPUT REGISTERED (i.e. 20R8)

IN THIS ARCHITECTURE MODE, PIN 1 = CK, PIN 13 = OE.

XOR(n)	OUTPUT POLARITY
0	ACTIVE LOW
1	ACTIVE HIGH

Registered Active High or Low Output

## ROW ADDRESS MAP DESCRIPTION

Figure 1 shows a block diagram of the row address map. There are a total of 44 unique row addresses available to the user when programming the GAL20V8. Row addresses 0-39 each contain 64 bits of input term data. This is the user array where the custom logic pattern is programmed. Row 40 is the Electronic Signature Word. It has 64 bits available for any user-defined purpose. Row 41-59 are reserved by the manufacturer and are not available to users.

Row 60 contains the architecture and output polarity information. The 82 bits within this word are programmed to configure the device for a specific application. Row 61 contains a one bit Security Cell that when programmed prevents further programming verification of the array. Row 63 is the row that is addressed to perform a bulk erase of the device, resetting it to a virgin state. Each of these functions is described in the following sections.

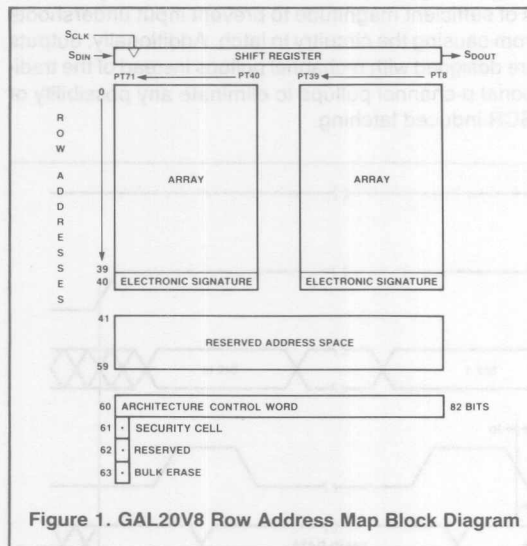


Figure 1. GAL20V8 Row Address Map Block Diagram

## ELECTRONIC SIGNATURE WORD

An Electronic Signature Word is provided with every GAL20V8 device. It resides at row address 40 and contains 64 bits of reprogrammable memory that can contain user-defined data. Some uses include ID codes, revision numbers, or inventory control. The ability to mark and identify parts electronically means improved parts handling and lower inventory costs. This signature data is always available to the user independent of the state of the Security Cell.

## ARCHITECTURE CONTROL WORD

All of the various configurations of the GAL20V8 are controlled by programming cells within the 82-bit Architecture Control Word that resides at row 60. The location of specific bits within the Architecture Control Word is shown in the control word diagram in Figure 2. The function of the SYN, AC0 and AC1(n) bits have been explained in the Output Logic Macrocell description. The eight polarity bits determine each output's polarity individually. The numbers below the XOR(n) and AC1(n) bits in Figure 2 show the output device pin numbers that the polarity bits control.

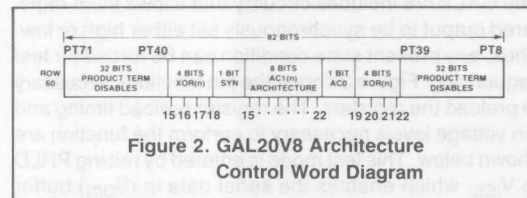


Figure 2. GAL20V8 Architecture Control Word Diagram

## SECURITY CELL

Row address 61 contains the Security Cell (one bit). The Security Cell is provided on all GAL20V8 devices as a deterrent to unauthorized copying of the array configuration patterns. Once programmed, the circuitry enabling array access is disabled, preventing further programming or verification of the AND array (rows 0-39). The cell can be erased only in conjunction with the array during a bulk erase cycle, so the original configuration can never be examined once this cell is programmed. Signature data is **always** available to the user.

## BULK ERASE MODE

By addressing row 63 during a programming cycle, a clear function performs a bulk erase of the array and the Architecture Control Word. In addition, the Electronic Signature Word and the Security Cell are erased. This mode resets a previously configured device to its virgin state.

## OUTPUT REGISTER PRELOAD

When testing state machine designs, all possible states and state transitions must be verified in the design, not just those required in the normal machine operations. This is because in system operation, certain events occur that may throw the logic into an illegal state (power-up, line voltage glitches, brown-outs, etc.). To test a design for proper treatment of these conditions, a way must be provided to break the feedback paths, and force any desired (ie. illegal) state into the registers. Then the machine can be sequenced and the outputs tested for correct next state conditions.

The GAL20V8 includes circuitry that allows each registered output to be synchronously set either high or low. Thus, any present state condition can be forced for test sequencing. Figure 3 shows the pin functions necessary to preload the registers. The register preload timing and pin voltage levels necessary to perform the function are shown below. This test mode is entered by raising PRLD to  $V_{IES}$ , which enables the serial data in ( $S_{DIN}$ ) buffer and the serial data out ( $S_{DOUT}$ ) buffer. Data is then serially shifted into the registers on each rising edge of the clock, DCLK. Only the macrocells with registered output configurations are loaded. If only 3 outputs have reg-

isters, then only 3 bits, need be shifted in. The registers are loaded from the bottom up, as shown in Figure 3.

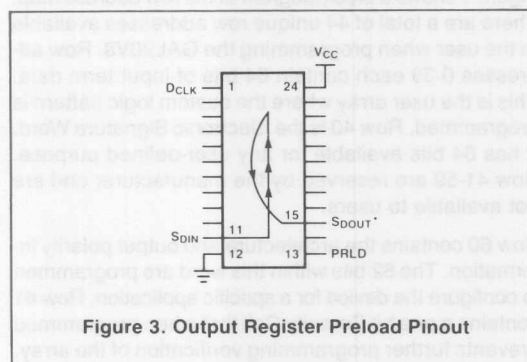


Figure 3. Output Register Preload Pinout

## LATCH-UP PROTECTION

GAL devices are designed with an on-board charge pump to negatively bias the substrate. The negative bias is of sufficient magnitude to prevent input undershoots from causing the circuitry to latch. Additionally, outputs are designed with n-channel pullups instead of the traditional p-channel pullups to eliminate any possibility of SCR-induced latching.

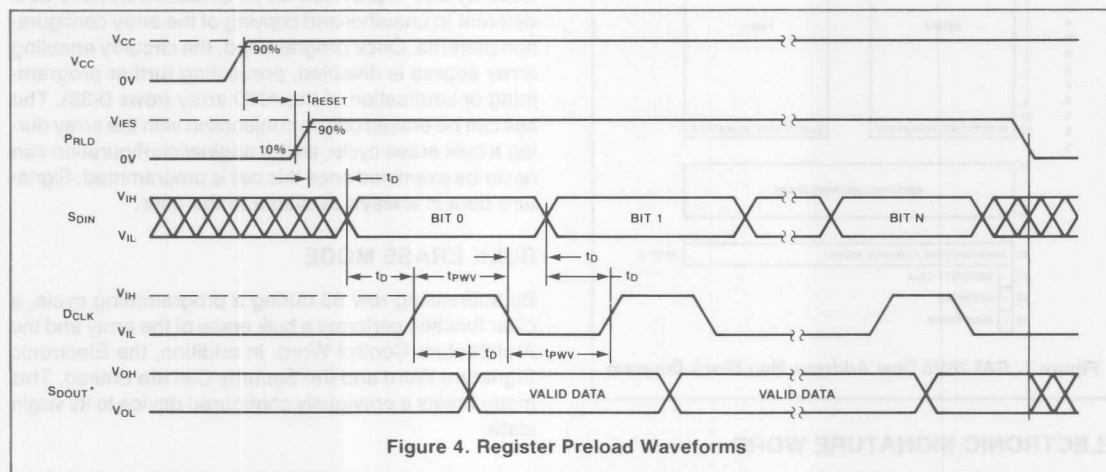
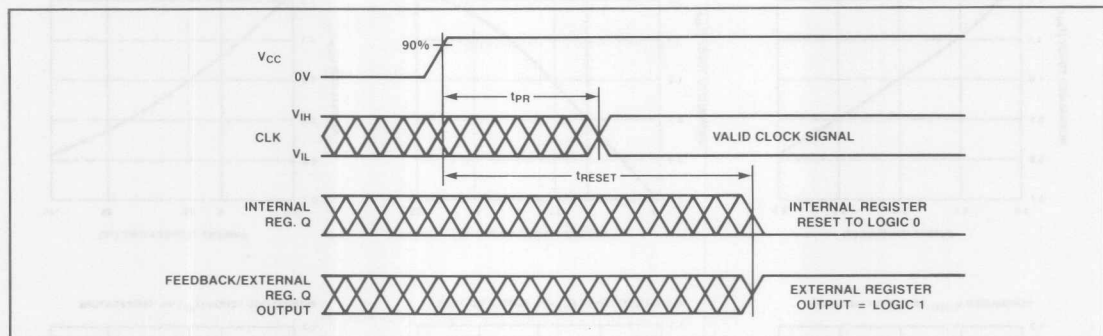


Figure 4. Register Preload Waveforms

\*NOTE — The  $S_{DOUT}$  output buffer is an open drain output during preload. This pin should be terminated to  $V_{CC}$  with a 10K resistor.

## POWER-UP RESET



Circuitry within the GAL20V8 provides a reset signal to all registers during power-up. All internal registers will have their Q outputs set low after a specified time ( $t_{RESET}$ ). As a result, the state on the registered output pins (if they are enabled through  $\overline{OE}$ ) will always be high on power-up, regardless of the programmed polarity of the output pins. This feature can greatly simplify state machine design by providing a known state on power-up.

The timing diagram for power-up is shown above. Because of the asynchronous nature of system power-up, some conditions must be met to guarantee a valid power-up reset of the GAL20V8. First, the  $V_{CC}$  rise must be monotonic. Second, the clock input must become a proper TTL level within the specified time ( $t_{PR}$ ). The registers will reset within a maximum of  $t_{RESET}$  time. As in normal system operation, avoid clocking the device until all input and feedback path setup times have been met.

2

## FIELD SUPPORT TOOLS

## PROGRAMMER/DEVELOPMENT SYSTEMS

VENDOR	SYSTEM	REVISION
DATA I/O	Model 29B	V04
	Logic Pak	
	P/T Adapter	V03
	303A-009	
	Model 60	**
STAG	PPZ/ZM2200	11/20
	ZL30 & ZL32	V30.41
	ZL30A	V30A.03
VARIX	Omni-Programmer	**
INLAB	Model 28	**
VALLEY DATA SCIENCES	Model 160	**

\*\* This version being qualified.

## SOFTWARE DEVELOPMENT TOOLS†

PACKAGE	VENDOR	REVISION
CUPL	Assisted Technology	2.1
ABEL	Data I/O	1.13
PLDtest	Data I/O	†
DASH-ABEL	Data I/O	1.0
PALASM	Monolithic Memories	†

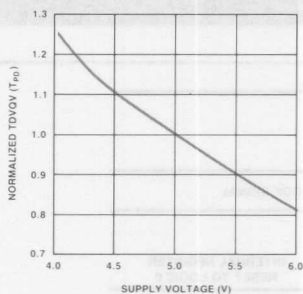
† When emulating PAL devices any revision of the software can be used to create the PAL JEDEC file. The programming hardware will automatically configure the GAL architecture.

Although it is possible to program GAL devices manually, LATTICE strongly recommends the use of approved programming hardware and software. Programming on unapproved equipment will generally void all guarantees.

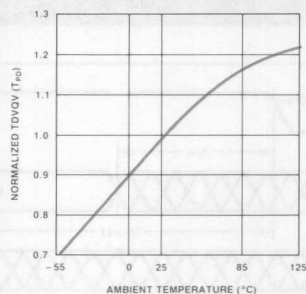
Approved equipment includes LATTICE programming algorithms that program the array, automatically configure the architecture control word, and track the number of program cycles each device has experienced (this information is stored within each GAL device). This in turn assures data retention and reliability. Contact the factory for specific conditions which must be met to gain programming equipment approval or for the current list of approved GAL programming equipment.



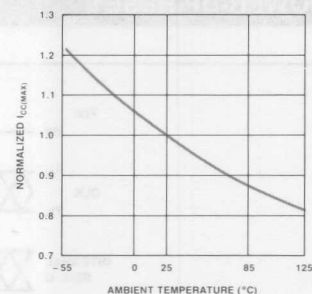
NORMALIZED TDQV ( $T_{PD}$ ) VS. SUPPLY VOLTAGE



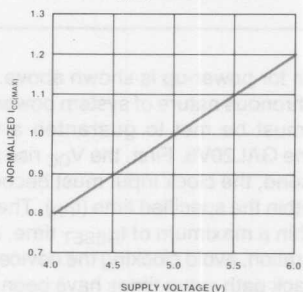
NORMALIZED TDQV ( $T_{PD}$ ) VS. TEMPERATURE



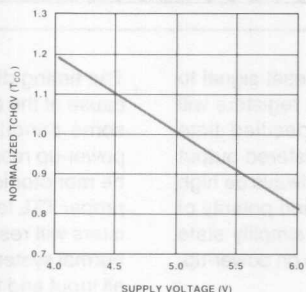
NORMALIZED  $I_{CC(MAX)}$  VS. TEMPERATURE



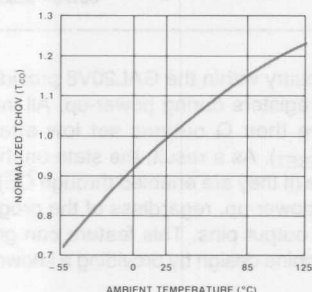
NORMALIZED  $I_{CC(MAX)}$  VS. SUPPLY VOLTAGE



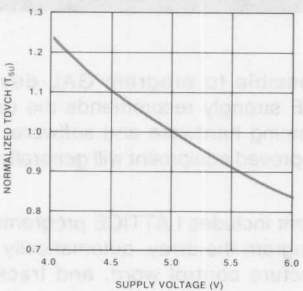
NORMALIZED TCHOV ( $T_{CO}$ ) VS. SUPPLY VOLTAGE



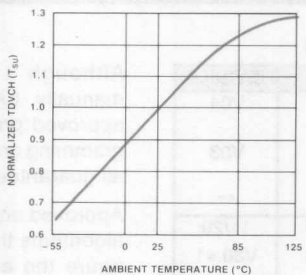
NORMALIZED TCHOV ( $T_{CO}$ ) VS. TEMPERATURE



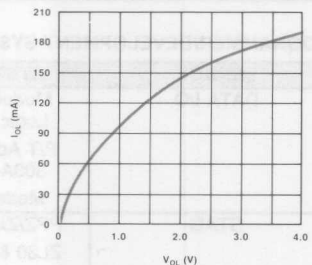
NORMALIZED TDVCH ( $T_{SU}$ ) VS. SUPPLY VOLTAGE



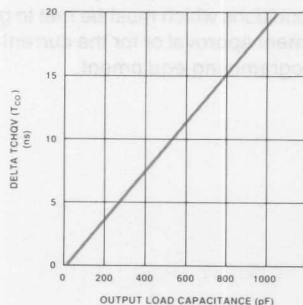
NORMALIZED TDVCH ( $T_{SU}$ ) VS. TEMPERATURE



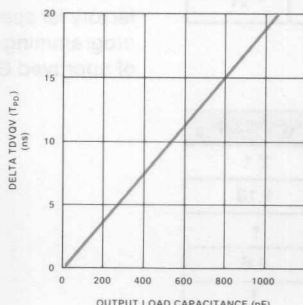
$I_{OL}$  VS.  $V_{OL}$



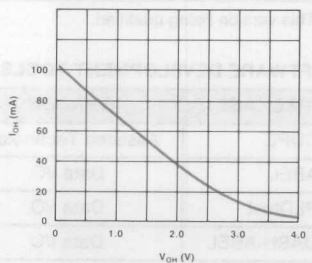
DELTA TCHOV ( $T_{CO}$ ) VS. OUTPUT LOADING



DELTA TDQV ( $T_{PD}$ ) VS. OUTPUT LOADING



$I_{OH}$  VS.  $V_{OH}$



All LATTICE Semiconductor products begin with exacting design criteria established for the devices' ultimate use in high-reliability programs. All products are manufactured, inspected, and tested in compliance with LATTICE's Quality Assurance program, which meets or exceeds all requirements as outlined in MIL-M 38510F Appendix A, as well as all inspection system requirements in MIL-I-45208A. LATTICE enforces all such requirements on any and all subcontractors involved in the manufacture of its product. LATTICE is solely responsible for securing and proving the documentation and control of its Quality Assurance program. All HI-REL(X) grade products undergo demanding screening procedures according to the Methods and

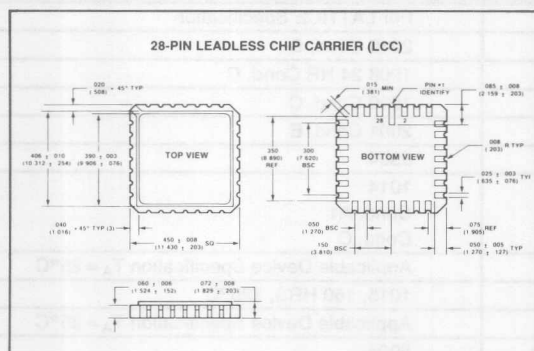
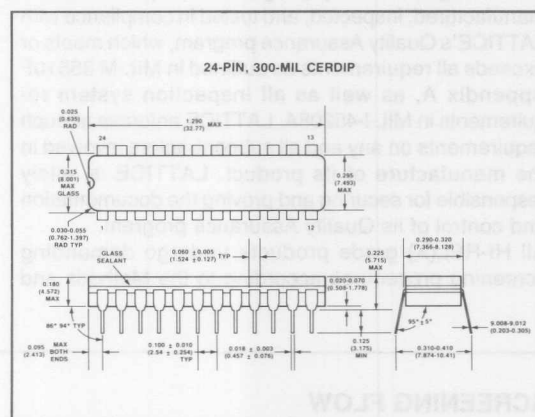
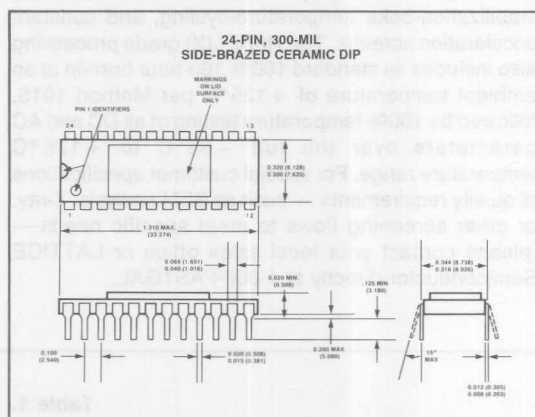
Requirements described below, which can be found in MIL-STD-883C, Method 5004. A high-reliability assembly flow is employed with 100% internal-visual, stabilization-bake temperature-cycling, and constant acceleration screens. The HI-REL(X) grade processing also includes as standard 100% 160-hour burn-in at an ambient temperature of +125°C per Method 1015, followed by 100% temperature testing of all DC and AC parameters over the full -55°C to +125°C temperature range. For special customer specifications of quality requirements — such as SEM analysis, X-ray, or other screening flows to meet specific needs — please contact your local sales office or LATTICE Semiconductor directly at 1-800-FASTGAL.

## SCREENING FLOW

Table 1.

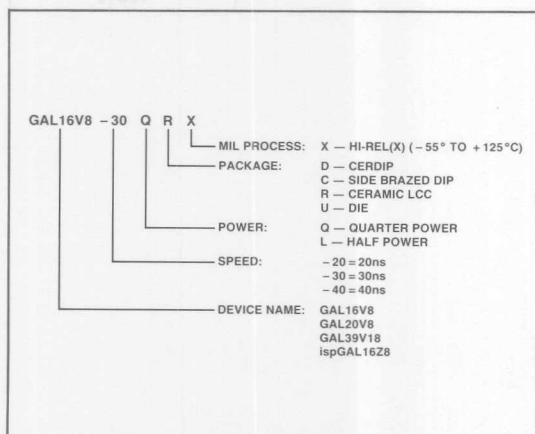
Screen	Method	Requirement
Wafer Lot Acceptance	Per LATTICE Specification	3 Wafers/Lot
Internal Visual	2010 Cond. B	100%
Stabilization Bake	1008 24 HR Cond. C	100%
Temp Cycling	1010 Cond. C	100%
Constant Acceleration	2001 Cond. E	100%
Visual Inspection	5004	100%
Hermeticity	1014	100%
Fine	Cond. A1	
Gross	Cond. C	
Pre-Burn-In Electrical	Applicable Device Specification $T_A = 25^\circ\text{C}$	100%
Burn-In	1015, 160 HRS, $125^\circ\text{C}$	100%
Post Burn-In Electrical	Applicable Device Specification $T_A = 25^\circ\text{C}$	100%
Percent Defective Allowable	5004	5%
Final Electrical Test	Applicable Device Specification $T_A = 125^\circ\text{C}$	100%
Final Electrical Test	Applicable Device Specification $T_A = -55^\circ\text{C}$	100%
OCI Sample Selection	MIL-M-38510F Section 4.5	Sample
External Visual	2009	100%

# PACKAGE INFORMATION



NOTE — All dimensions are in inches and parenthetically in millimeters. Inch dimensions govern.

## ORDERING INFORMATION



The specifications and information contained herein are subject to change without notice.

## SPEED/POWER CROSS-REFERENCE GUIDE

SPEED	POWER	GAL DEVICE	BIPOLAR PAL DEVICE
15ns	45mA	–15Q	—
15ns	90mA	–15L	—
15ns	180mA	use –15L	B
* 20ns	50mA	–20Q	—
* 20ns	90mA	–20L	—
* 20ns	210mA	use –20L	B MIL
25ns	45mA	–25Q	—
25ns	90mA	–25L	B-2
25ns	180mA	use –25L	A
* 30ns	50mA	–30Q	—
* 30ns	90mA	–30L	B-2 MIL
* 30ns	210mA	use –30L	A MIL
35ns	45mA	–35Q	B-4
35ns	90mA	–35L	A-2
35ns	180mA	use –35L	STD
* 40ns	50mA	–40Q	B-4 MIL
* 40ns	90mA	–40L	A-2 MIL
* 40ns	210mA	use –40L	STD MIL

\* MILITARY TEMPERATURE RANGE

## Logic Fundamentals

Foundation to the digital logic design process is an understanding of the basics of Boolean algebra. Those readers desiring a more complete review or education on these concepts are referred to the many fine books on the topic listed as the references at the end of this chapter. This section deals with the fundamentals of Boolean algebra necessary to implement a basic logic function in a PLD.

## Boolean Algebra

Boolean theory comes to us from George Boole and his 1854 publication, 'An Investigation of the Laws of Thought.' The conditions of yes or no, true or false, high or low, etc. is a Boolean condition. These 'black or white' conditions are all around us.

For example, let's look at the alarm clock that you use this morning. Its functionality can be defined as a Boolean function.

If the alarm is not set, it will not sound.  
If the alarm is set and the time matches the preset alarm time, it will buzz.

This functionality can be expressed in a table format (Table 1). Where 0 stands for 'false' or 'no', and 1 stands for 'true' or 'yes'.

Notice that only the combination of variables where both 'Alarm Set' and 'Time Match' are true results in a sound of one (or buzz).

The basic functionality defined above can also be written in an equation as follows:

$$\text{Sound} = \text{Set} \cdot \text{Match}$$

The equation is read as 'The alarm sounds only when the alarm is set AND the time matches.' This implements the logical AND function.

## Basic Functions

Boolean algebra is performed on the set of {True, False} which is commonly abbreviated as {1,0}. All Boolean operations are performed on variables having only these two states and all Boolean results are expressed in terms of one of these two states. The functionality of Boolean algebra for outweights its apparent simplicity, as we will see in the coming sections.

The previous equation used the Boolean AND function, which is one of the three basic Boolean functions: AND, OR and NOT. All three are readily implemented

Table 1. Alarm Clock Functionality

Alarm Set	Time Match	Sound
0	0	0
0	1	0
1	0	0
1	1	1

Lattice Semiconductor offers a family of high-performance E<sup>2</sup>CMOS programmable logic devices that provides enhanced performance for both existing and future logic designs. Known as GAL (Generic Array Logic) devices, they offer a powerful new architecture that is fully compatible with existing logic development tools.

The design of logic systems has long been faced with newer and better ways to design systems. PLD, TTL, PAL, gate array, custom . . . Each of these 'alternatives' in the design process has solved some aspect of the design problem, while changing yet another series of problems in the future.

INTRODUCTION	1
GAL DEVICE SPECIFICATIONS	2
LOGIC TUTORIAL	3
USING DEVELOPMENT TOOLS	4
GAL DEVICE APPLICATIONS	5
TECHNICAL BRIEFS	6
E <sup>2</sup> CMOS TECHNOLOGY OVERVIEW	7
GAL DEVICE QUALITY AND RELIABILITY	8
ARTICLE REPRINTS	9
APPENDICES	10
SALES OFFICES	11

DESCRIPTION

Lattice Semiconductor offers a family of high-performance E<sup>2</sup>CMOS programmable logic devices that provides enhanced performance for both existing and future logic designs. Known as GAL (Generic Array Logic) devices, they offer a powerful new architecture that is fully compatible with existing logic development tools.

The designer of logic systems has long been faced with newer and better ways to design systems: RTL, TTL, PAL, gate array, custom... Each of these 'enhancements' to the design process has solved some aspect of the design problem, while bringing yet another series of problems to the table.

This section provides a brief overview of the fundamentals necessary to design with GAL devices. Experienced users of PLDs can skip directly to page 3-10, where the enhanced architecture of GAL devices is discussed.

Logic Fundamentals

Fundamental to the digital logic design process is an understanding of the basics of Boolean algebra. Those readers desiring a more complete review or education on these concepts are referred to the many fine books on the topic (also see the references at the end of the chapter). This section deals with the fundamentals of Boolean algebra necessary to implement a basic logic function in a PLD.

Boolean Algebra

Boolean theory comes to us from George Boole and his 1854 publication 'An Investigation of the Laws of Thought.' The condition of yes or no, true or false, high or low, etc. is a Boolean condition. These 'black or white' conditions are all around us.

For example, let's look at the alarm clock that got you up this morning. Its functionality can be defined as a Boolean function.

- a) If the alarm is not set, it will not sound.
- b) If the alarm is set and the time matches the preset alarm time, it will buzz.

This functionality can be expressed in a table format (Table 1), where 0 stands for 'false' or 'off,' and 1 stands for 'true' or 'on'.

Notice that only the combination of variables where both 'Alarm Set' and 'Time Match' are true results in a sound of one (or buzz).

The basic functionality defined above can also be written in an equation as follows:

$$\text{Sound} = \text{Set} * \text{Match}$$

The equation is verbalized as 'The alarm sounds only when the alarm is set AND the time matches.' The '\*' represents the logical AND function.

Basic Functions

Boolean algebra is performed on the set of [True, False] which is commonly abbreviated as [1,0]. All Boolean operators are performed on variables having only these two states and all Boolean results are expressed in terms of one of these two states. The functionality of Boolean algebra far outweighs its apparent simplicity, as we will see in the coming sections.

The previous equation used the Boolean AND function, which is one of the three basic Boolean functions: AND, OR and NOT. All three are readily implemented

Table 1. Alarm Clock Functionality

Alarm Set	Time Match	Alarm Sounds
0	0	0
0	1	0
1	0	0
1	1	1



with transistor circuits. For example, electronic implementations of the OR function are shown in schematic form in Figure 1. The differences in transistor technology between bipolar and CMOS have no bearing on the logical 1 or 0 result of the OR operation. The logical operation of the OR function is shown in the top entries of Figure 2. The 'truth' table shows that C is true when either A OR B is true. Only one of the operands needs be true for the result to be true; however, if both operands are true, the result remains true. (Remember that this is a logical function and not an arithmetic function.)

The AND operator works in a similar fashion. For the result to be true with an AND function, all of the operands must be true. Thus, C is true only when both A AND B are true.

The NOT operator is the simplest of the three operators; however, it plays an essential role in manipulating

complex data. The NOT function is simply an inversion: when the input is true, the output is false; when the input is false, the output is true.

The three Boolean operators are accorded specific priorities, or precedence, when evaluating equations. Functions in parentheses are always evaluated first. The NOT operator is evaluated next, before the AND, which is applied before the OR. As an example, examine the following equations:

$$(1) C = \bar{A} + \bar{B}$$

$$(2) D = (\bar{A} + \bar{B})$$

In case 1, the individual values of A and B are inverted before the OR function is evaluated. In case 2, however, the parentheses demand that the OR be evaluated first, and then the result is inverted by the NOT.

Boolean algebra also has the familiar Associative and Commutative theories for the AND and OR operators, whereby equations can be expanded or reduced through the laws of Associativity and Commutativity. The Associative theory specifies that like operations can be evaluated in any order, regardless of parenthesis:

$$(A + B) + C = A + (B + C)$$

The theory requires that the operators be the same; thus the following inequality:

$$(A + B) * C \neq A + (B * C)$$

The Commutative theory specifies that, with like operators, the order of the variables (or operands) is not important; thus:

$$A * B = B * A$$

### Basic Theory

The theory of Boolean algebra is based on several postulates that are given to be true. The proofs and analyses of the postulates will not be presented here; however, you may wish to refer to the reference material for further details.

Figure 1. OR Gate Circuits

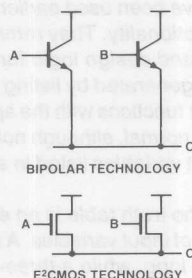





Figure 2. Boolean Operators

LOGIC SYMBOL	TRUTH TABLE	EQUATION	FUNCTION	LOGIC OPERATOR															
	<table><tr><th>A</th><th>B</th><th>C</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	C	0	0	0	0	1	1	1	0	1	1	1	1	$C = A + B$	OR	"+"
A	B	C																	
0	0	0																	
0	1	1																	
1	0	1																	
1	1	1																	
	<table><tr><th>A</th><th>B</th><th>C</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	C	0	0	0	0	1	0	1	0	0	1	1	1	$C = A \cdot B$ or $C = A * B$ or $C = AB$	AND	"*" or "•"
A	B	C																	
0	0	0																	
0	1	0																	
1	0	0																	
1	1	1																	
	<table><tr><th>A</th><th>C</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	A	C	0	1	1	0	$C = \bar{A}$ $C = /A$	NOT	"/'" or "¯"									
A	C																		
0	1																		
1	0																		

Also necessary for a full understanding of Boolean algebra is a discussion of the details of equation manipulation, minimization, and expansion. The recent advances in PLD software tools now allow the software to handle all of the reduction automatically. As such, we spend only a few brief words on this topic.

### Postulates

The postulates, as proposed by Huntington in 1904 (see reference (3) at the end of this section) are as follows:

#### 1) Additive and Multiplicative Identity Elements

- (a)  $x + 0 = x$
- (b)  $x \cdot 1 = x$

#### 2) Commutativity

- (a)  $x + y = y + x$
- (b)  $x \cdot y = y \cdot x$

#### 3) Distributivity

- (a)  $x + (y \cdot z) = (x + y) \cdot (x + z)$
- (b)  $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$

#### 4) Complementation

- (a)  $x + \bar{x} = 1$
- (b)  $x \cdot \bar{x} = 0$

Notice that each of the postulates is really a set of two definitions that reflects the action of the two basic algebraic operators, AND and OR.

The Duality Principle takes advantage of the parallels in the effect of the AND and OR operators. Duality is used extensively to generate the second part of the set of theorems below.

Duality states that if the following replacements are made, a logical equivalent expression can be generated:

- (a) Replace all Trues with Falses
- (b) Replace all Falses with Trues
- (c) Replace all ANDs with ORs
- (d) Replace all ORs with ANDs

### Theorems

Derived from the postulates and the Duality Principle are a set of theorems that allows for simplified analysis of a logic equation:

#### 1) Idempotence

- (a)  $x \cdot x = x$
- (b)  $x + x = x$

#### 2) Special properties of 0 and 1

- (a)  $x \cdot 0 = 0$
- (b)  $x + 1 = 1$
- (c)  $\bar{0} = 1$
- (d)  $\bar{1} = 0$

#### 3) Absorption

- (a)  $x \cdot (x + y) = x$
- (b)  $x + (x \cdot y) = x$

#### 4) Associativity

- (a)  $x + (y + z) = (x + y) + z$
- (b)  $x \cdot (y \cdot z) = (x \cdot y) \cdot z$

#### 5) If $x \cdot y = y$ and $x + y = y$ then $x = y$

#### 6) $\overline{(\bar{x})} = x$

#### 7) DeMorgan's Law

- (a)  $\overline{(x + y)} = \bar{x} \cdot \bar{y}$
- (b)  $\overline{(x \cdot y)} = \bar{x} + \bar{y}$

### Truth Tables, Karnaugh Maps

Truth tables have been used earlier in this section to describe logic functionality. They remain as a powerful tool to document and design logic functions.

A truth table is generated by listing all combinations of the various input functions with the appropriate output logic function. The normal, although not required, format is to have the input variables listed in a binary counting sequence.

The length of the truth table is an exponential function of the number of input variables. A table of two-input functions is  $2^2 = 4$  long, while a three-input function is  $2^3 = 8$  long.

A Karnaugh map is a visual tool that aids in the reduction of logic functions to either of two special formats that are easily transferred into a PLD logic map. These formats — Sum of Products and Product of Sums — are discussed in a subsequent section.

Since a thorough understanding of Karnaugh maps is not required to use programmable logic devices, (especially since the current-generation development software tools will automatically and accurately reduce and minimize input equations), the topic is not discussed in this book.

Some of the older software development packages that are assembler-based, however, do require that fully minimized equations be used as the input functions, and the reader with access to only such tools will likely need to fully familiarize himself with Karnaugh-map minimization techniques.

### Equations

A programmable device requires that a particular format of data be used to define the final functionality of

the device. It is the goal of the designer or the development software to transform the logic definition into an acceptable format. The Product of Sums (POS) format can be used to describe any combinational logic function. This two-level format consists of logical OR terms that are ANDed together. Thus,

$$(1) y = a \cdot (c + d) + b \cdot c + b \cdot d$$

can be simplified to:

$$(2) y = a \cdot (c + d) + b \cdot (c + d)$$

$$(3) = (a + b) \cdot (c + d)$$

which is an AND of sum terms.

The most common representation is the dual of the product-of-sums format and is known as the Sum of Products. The basic PLD array interconnects are of this form.

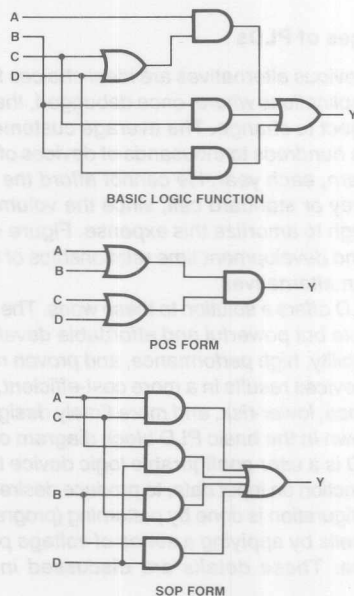
The Sum of Products (SOP) consists of several AND terms ORed (summed) together. Eq.(1) can also be simplified to a SOP form:

$$(4) y = a \cdot (c + d) + b \cdot c + b \cdot d$$

$$(5) = a \cdot c + a \cdot d + b \cdot c + b \cdot d$$

The above transformations are shown in Figure 3.

Figure 3. Basic Function Formats



## DeMorgan's Law

A closer examination of the above two implementations of the same function in POS (Eq.(3)) and SOP (Eq.(5)) forms shows that the number of terms feeding into the final gate varies with the implementation. The SOP format required 4 terms, while the POS format required only 2 terms. This observation is critical, since the total number of terms in any PLD is limited.

DeMorgan's Law (defined earlier in this section) is a simple rule that can quickly convert from SOP to POS or back, without altering the final logic function. This can often allow the number of terms to be reduced by as much as 50%, to overcome device limitations. Eq.(3) can be converted to SOP as follows :

$$(3) y = (a + b) \cdot (c + d)$$

using the duality principle:

$$(6) y = (\bar{a} \cdot \bar{b}) + (\bar{c} \cdot \bar{d})$$

and then gathering the NOT functions to the left side of the equal sign:

$$(7) \bar{y} = (\bar{a} \cdot \bar{b}) + (\bar{c} \cdot \bar{d})$$

Eq.(7) is the SOP form. Note, however, that the output function is inverted (or 'active low'). A subsequent inversion function will be required to produce the original output function.

## Canonical Form

The canonical is a 'long-form' representation of logic equations that, technically, represents an equation as the sum of its minterms. (A minterm is a unique representation of input variables and would represent any one row in our truth tables.) Canonical forms are not more compact; rather, they provide a simple means of converting equations from POS to SOP and vice-versa.

Again, the use of canonical forms is required only by some of the previous-generation software packages and has a limited practical value in designing PLDs.

## Reduction of Equations

Generally a complex logic function must be represented in a specific and reduced format to be implemented into a PLD. The various methods touched on above — Karnaugh maps, DeMorgan's Law, and Canonical forms — are used to manipulate the equations in conjunction with the basic Postulates and Theorems.

Current-generation software handles all equation minimization and will actually allow the use of higher-

level functions, such as state description and macro functions, as input functions.

## PLD ALTERNATIVES

It is not uncommon for a system to be designed using digital logic devices from any of several basic categories, exploiting the advantages and designing around the disadvantages of each category. The basic groups are:

- Standard product: TTL, LSTTL, etc. (SSI & MSI devices)
- Software-configurable LSI Devices (microprocessors)
- Application-Specific ICs (ASICs)

Though the most widely available, standard products are designed to meet the general needs of a large customer base. The user must prepare for a high degree of interconnect, and design around the various stand-alone-device performance and functional specifications. The result of using standard products can be higher system cost, higher unit count, and increased power-supply and board-space requirements. This alternative may also result in lower system reliability, due to the high number of components and interconnects. The advantage lies in the high performance (at the cost of power) obtainable using SSI and MSI devices.

Software-configurable microprocessor devices have an inherent flexibility and customizability that is difficult to match with other device types; however, these devices typically operate at one-tenth the speed of dedicated ICs, and cannot serve directly in the data path of speed-critical applications. In addition, it is not uncommon for a microprocessor to require an 'army' of support chips to interface to a real-world application.

The devices in the ASIC category offer advantages over other alternatives, in that they perform a function defined by the user that is optimized for his application. The ASIC category further divides into three categories of functionally specific devices: Standard Cells, Gate Arrays, and Programmable Logic Devices (PLDs).

The standard-cell approach uses pre-configured, pre-tested and pre-characterized logic blocks to construct a custom silicon chip for the designer. The chip is usually designed by hand, using a graphics terminal. The outcome is a fairly efficient logic design that may take weeks or months to complete, while incurring a hefty up-front engineering charge, or non-recurring expense (NRE). In addition, the custom piece of silicon will take additional weeks, if not months, to manufacture.

The development time and complexity of a standard-cell design severely impact the ability to incorporate changes or corrections to the design. In addition, the

customer is typically obligated for some minimum production lot size to cover the manufacturer's expenses. Since most logic design is subject to revision during the debug phase, the time and dollar penalties of the standard cell approach make this a relatively high risk.

The gate-array approach has gained extensive market recognition as a more optimal 'bridge' between the standard cell (full custom) and programmable alternatives. The gate array is a pre-manufactured silicon matrix that awaits only a custom interconnect pattern to establish functionality. The designer can choose from NAND gates, flip-flops, and various types of buffers to construct his logic.

The flexibility of a gate array is less than that of a standard-cell device, since the user must interconnect existing structures. However, since the device usually has many pins — 68 or more — it offers greater logic functionality than a typical PLD. This increased functionality, however, comes at the penalty of lower performance.

Since the gate array relies on only one or two custom mask layers, the wafer fabrication can be done much more quickly than with the standard cell. The turn time from design completion to final chip for a gate array is, at best, 4 to 8 weeks. Although not as costly up front as a standard cell, there is still an up-front development cost, a minimum lot size, and a long and costly cycle for logic changes. These features make this, too, a risky approach.

## Advantages of PLDs

The previous alternatives are ideal choices for high-volume applications where, once debugged, the design is not subject to change. The average customer, however, uses hundreds to thousands of devices of a given logic pattern, each year. He cannot afford the NRE of a gate array or standard cell, since the volume is not high enough to amortize this expense. Figure 4 shows the cost and development time relationships of the various design alternatives.

The PLD offers a solution to these woes. The low unit cost, simple but powerful and affordable development tools, flexibility, high performance, and proven reliability of these devices results in a more cost-efficient, higher-performance, lower-risk, and more timely design cycle.

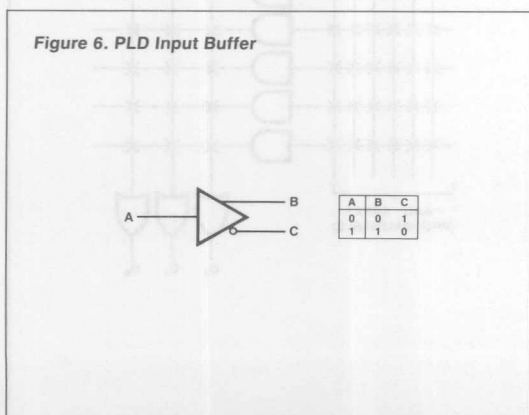
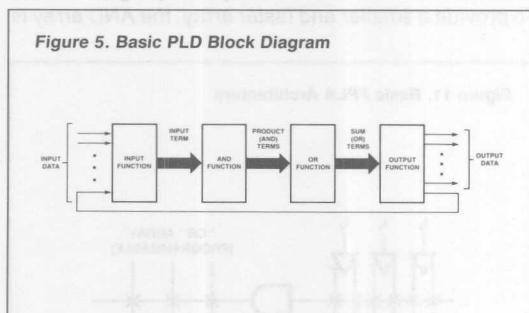
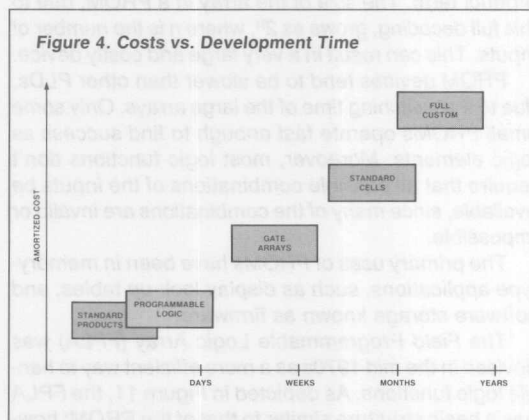
As shown in the basic PLD block diagram of Figure 5, the PLD is a user-configurable logic device that performs a function on input data, to produce desired output data. Configuration is done by patterning (programming) memory cells by applying a series of voltage pulses to the device. These details are discussed in a later section.

As one would expect, there are many types of PLD's, each optimized to perform a specific logic function with



given performance criteria. Before we look at the various types of PLDs, we must review a few logic conventions used to describe (and illustrate) these devices.

Figure 6 shows a typical PLD input buffer. Its two outputs are the true and complement of the input, as shown by the truth table. Figure 7 illustrates the convention used to reduce the complexity of a logic diagram without sacrificing any of the clarity. The traditional representation



of an AND gate shows three inputs: A, B, and C. The PLD representation has the same three inputs. This shorthand reflects the three distinct input terms of the prior drawing. The structure of a multiple-input AND gate is known as a Product Term.

Referring to Figure 8, we see that the solid-dot connection in the previous figure represents a permanent connection. A programmable interconnection would appear as an X over the intersection, as shown. The X implies that the connection is intact, whereas the absence of an X implies no interconnection.

Figure 9 details the default conditions for AND gates. From the diagram, you can see that the AND gate for output D is connected to all the input terms. The equation for D is

$$D = A \cdot \bar{A} \cdot B \cdot \bar{B}$$

which can be simplified using Boolean algebra

$$D = (A \cdot \bar{A}) \cdot (B \cdot \bar{B})$$

$$D = (0) \cdot (0)$$

$$D = 0$$

The connection of both the true and complement of a given input buffer to a single product term results in that product term always being a logic 0.

A shorthand notation for leaving all of the input buffers connected is illustrated on output E. Since logic-

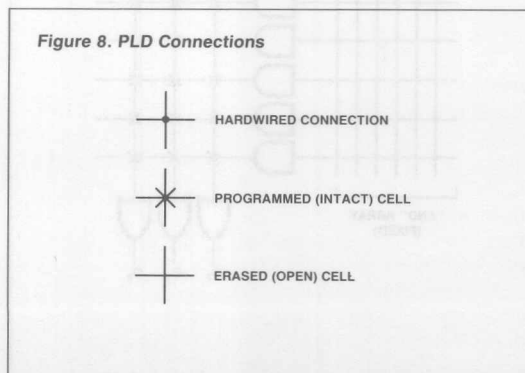
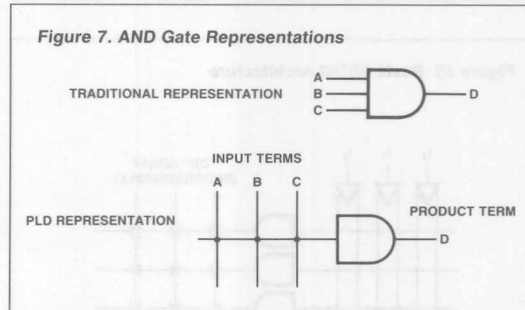




diagram maps are usually supplied without any of the connections shown as intact, it is much simpler for the designer to connect a whole product term (the device default) by simply drawing the X within the AND gate. Again, this product term will always be a logic 0.

Output F, in contrast, does not have any input terms connected to its product term. This product term will always 'float' to a logic 1, resulting in a 1 on the output. In the following sections, where various PLD architectures will be examined in detail, we will see why this design practice is not recommended.

Figure 9. AND Gate Default Conditions

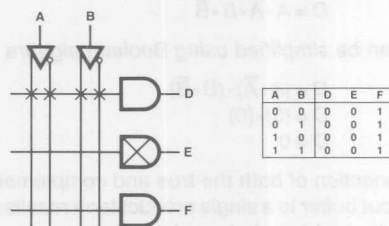
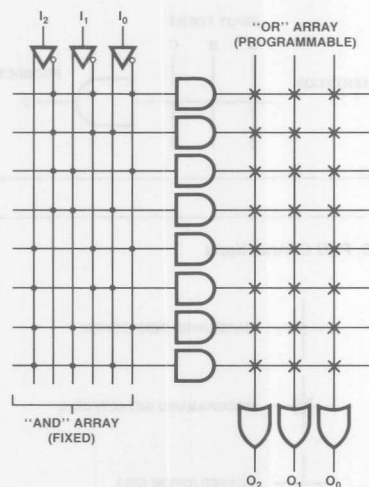


Figure 10. Basic PROM Architecture



## PROMs and FPLAs

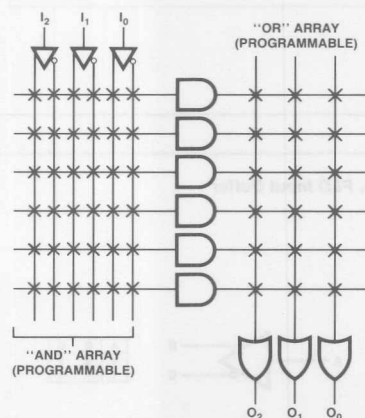
The first PLDs were made available in the early 1970s. Known as Programmable Read-Only Memories (PROMs), these devices can now be purchased in densities ranging from 64 to 1 million bits. They comprise a programmable OR array fed by a fixed AND array (Figure 10). The AND array is a 'fully decoded' array, meaning that all possible combinations of the inputs  $I_0...I_2$  have a unique product term. The size of the array in a PROM, due to this full decoding, grows as  $2^n$ , where  $n$  is the number of inputs. This can result in a very large and costly device.

PROM devices tend to be slower than other PLDs, due to the switching time of the large arrays. Only some small PROMs operate fast enough to find success as logic elements. Moreover, most logic functions don't require that all possible combinations of the inputs be available, since many of the combinations are invalid or impossible.

The primary uses of PROMs have been in memory-type applications, such as display look-up tables, and software storage known as firmware.

The Field Programmable Logic Array (FPLA) was devised in the mid 1970s as a more efficient way to handle logic functions. As depicted in Figure 11, the FPLA has a basic structure similar to that of the PROM; however, both its AND and OR arrays are programmable. To provide a smaller and faster array, the AND array is

Figure 11. Basic FPLA Architecture



not fully decoded. Product terms can be shared by any or all of the OR terms.

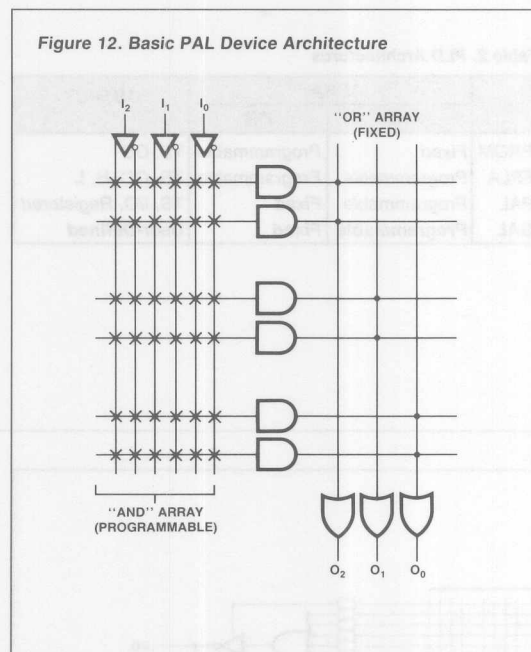
In the following example, you can see that the functions require 7 product terms, of which 5 are unique:

$$O_0 = I_0 * I_1 * I_2 + \bar{I}_1 * I_2$$

$$O_1 = I_0 * I_1 * I_2 + \bar{I}_0 * \bar{I}_1 * \bar{I}_2 + I_0 * \bar{I}_1 * I_2$$

$$O_2 = \bar{I}_0 * \bar{I}_1 * \bar{I}_2 + I_1 * I_2$$

Figure 12. Basic PAL Device Architecture



This function can be implemented in our FPLA device, since the six available product terms can accommodate the five unique product terms of the example.

With its smaller arrays, an FPLA operates faster than a PROM. FPLAs find extensive usage in applications where the output functions are very similar, allowing full utilization of the shared product terms. The dual programmable arrays make the designer's task easier, since he can control all of the functionality of the device. Of commercially available devices, design engineers have objected in the past to the quality of the software support, the unavailability of programming tools, the device cost, and the speed with which these devices processed data. Great strides are being made on all of these fronts, however, and you can expect to see a revival of enthusiasm in FPLA types of devices.

### Enter PAL Devices

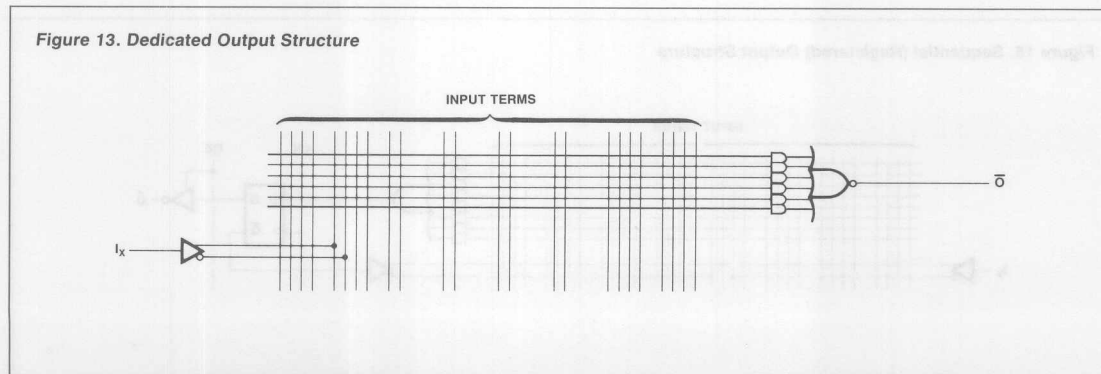
The PAL approach of the late 70s, illustrated in Figure 12, again varies the array control. This time, the AND array is programmable and the OR array is fixed. This approach offers the highest performance and the most efficient architecture for most logic functions.

The quantity of product terms per output is fixed by the hardwired OR array. The typical logic function requires some 3 to 4 product terms, well under the 7 to 8 available on current-generation devices.

PAL device architectures — the number of inputs, outputs and product terms — have been fixed by the manufacturer, based on a guess as to what the designer may ultimately want. However, the dozens of device types introduced over the last 8 years essentially offer various permutations of three basic output structures.

The first is shown in Figure 13, which illustrates an input and an output with six product terms. The output is always enabled and active-low (notice the invert at the OR gate). The true and complement of the input are available to the AND array.

Figure 13. Dedicated Output Structure



The second output structure is actually an I/O pin, shown in Figure 14. The output logic is an active-low function of seven product terms. The data from the pin also feeds back into the AND array.

Notice that the output buffer is controlled by its own product term, allowing dynamic I/O control. This dynamic control can be used either to determine the ratio of device inputs to outputs, or to disable the outputs when connected in a bus environment.

Third is a sequential, or registered output, shown in Figure 15. The logic OR of eight product terms is available to the designer. Here, the register state (both true and complement) feeds back into the array, as well as into an output buffer with a bank-controlled output-enable feature. The clock is common, as well, minimizing switching skew between buffers, and the register is a high-speed D type. The feedback of the register data into the AND array allows the current-state data (in the registers) to be part of the next-state function. This is necessary for most sequential functions, such as counting and shifting operations.

#### Meet GAL Devices

The architectures of the three types of programmable logic devices discussed so far are summarized

in Table 2. The Lattice Semiconductor GAL device is listed as a fourth element, since its architecture is a next-generation enhancement of the basic PAL architecture and deserves special review.

The GAL device was conceived to provide enhanced functionality, quality, design support, and flexibility without compromising performance.

The GAL architecture is the now-familiar programmable AND array driving a fixed OR array. The difference

Table 2. PLD Architectures

	ARRAY		OUTPUT
	AND	OR	
PROM	Fixed	Programmable	TS, OC
FPLA	Programmable	Programmable	TS, OC, H, L
PAL	Programmable	Fixed	TS, I/O, Registered
GAL	Programmable	Fixed	User-Defined

Figure 14. Asynchronous I/O Structure

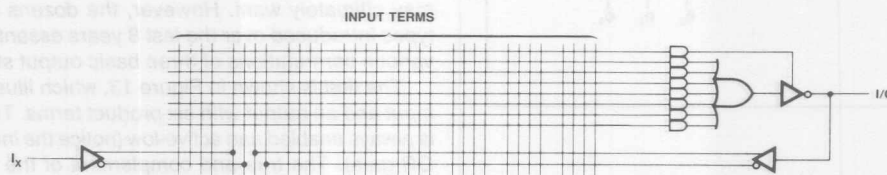


Figure 15. Sequential (Registered) Output Structure

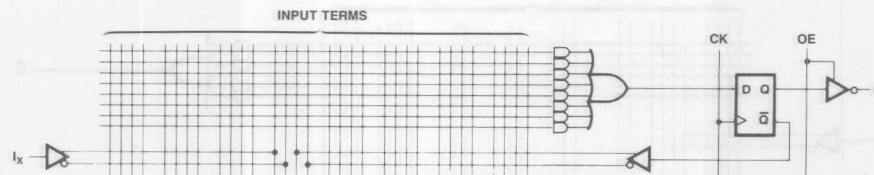
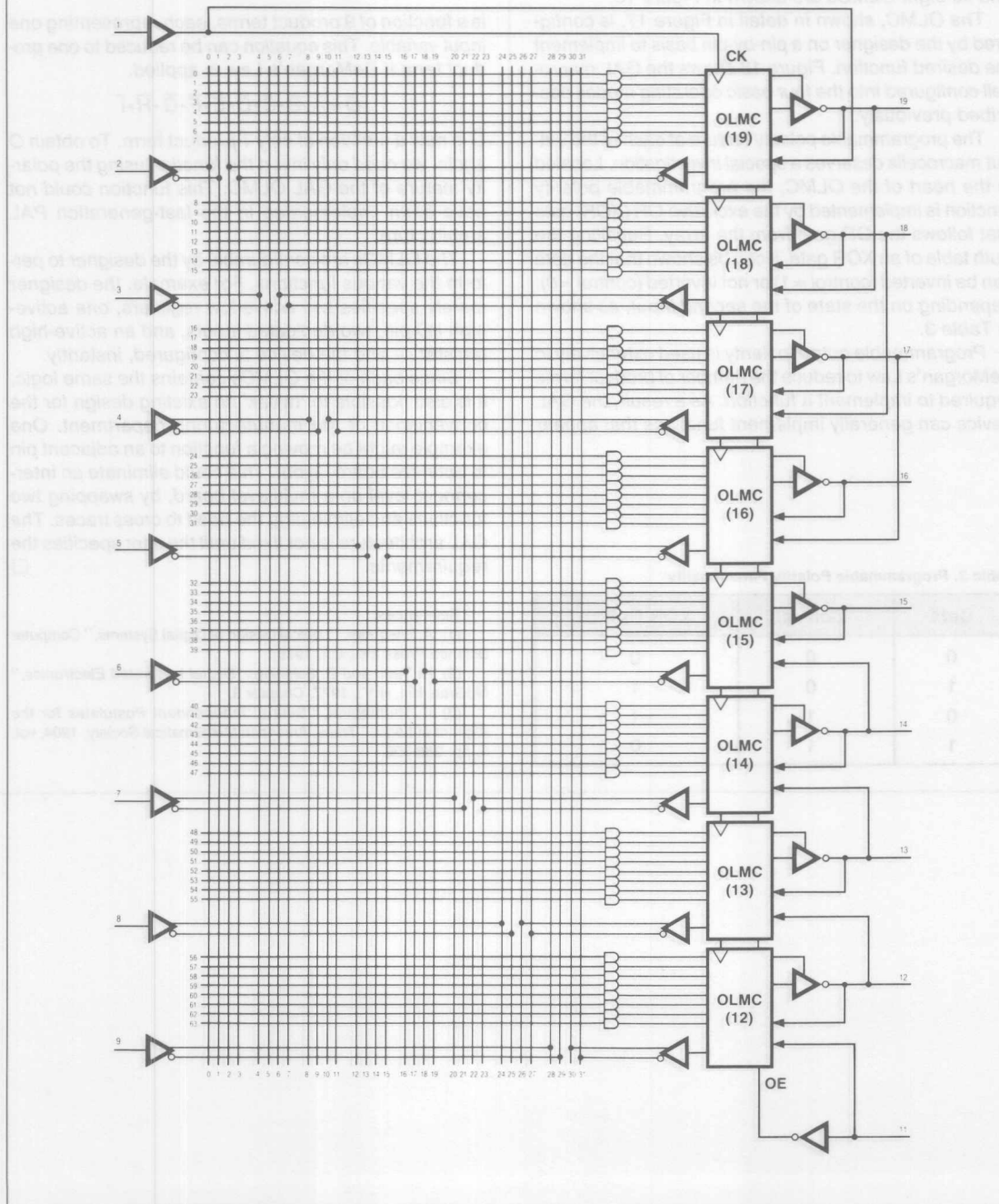


Figure 16. GAL16V8 Block Diagram



is in the architecture and flexibility of the output functions. The GAL device integrates an Output Logic Macro-Cell (OLMC) on each of its output pins. The GAL16V8 and its eight OLMCs are shown in Figure 16.

The OLMC, shown in detail in Figure 17, is configured by the designer on a pin-by-pin basis to implement the desired function. Figure 18 shows the GAL macro-cell configured into the four basic operating modes described previously.

The programmable polarity feature of each of the output macrocells deserves a special investigation. Located in the heart of the OLMC, the programmable polarity function is implemented by the exclusive-OR (XOR) gate that follows the OR gate from the array. Recalling the truth table of an XOR gate, it can be shown that the data can be inverted (control = 1) or not inverted (control = 0), depending on the state of the second input, as shown in Table 3.

Programmable output polarity is used extensively in DeMorgan's Law to reduce the number of product terms required to implement a function. As a result, the GAL device can generally implement functions that appear

to require more than 8 product terms per output. For example,

$$O = A + B + C + D + E + F + G + H + I$$

is a function of 9 product terms, each representing one input variable. This equation can be reduced to one product term if DeMorgan's Law is applied.

$$\overline{O} = \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \overline{D} \cdot \overline{E} \cdot \overline{F} \cdot \overline{G} \cdot \overline{H} \cdot \overline{I}$$

$\overline{O}$  is now a function of only 1 product term. To obtain  $O$  again, we need only invert the function using the polarity feature of the GAL OLMC. This function could not have been implemented in the last-generation PAL architecture.

The OLMCs are configurable by the designer to perform the various functions. For example, the designer merely specifies two active-low registers, one active-high I/O pin, two dedicated inputs, and an active-high register — and the device is configured, instantly.

Since each of the OLMCs contains the same logic, it is also possible to 'tweak' an existing design for the convenience of the manufacturing department. One example might be moving a function to an adjacent pin to optimize board layout. This could eliminate an interconnect level on a multilevel board, by swapping two functions and eliminating the need to cross traces. The GAL architecture is not fixed until the user specifies the requirements. □

Table 3. Programmable Polarity Functionality

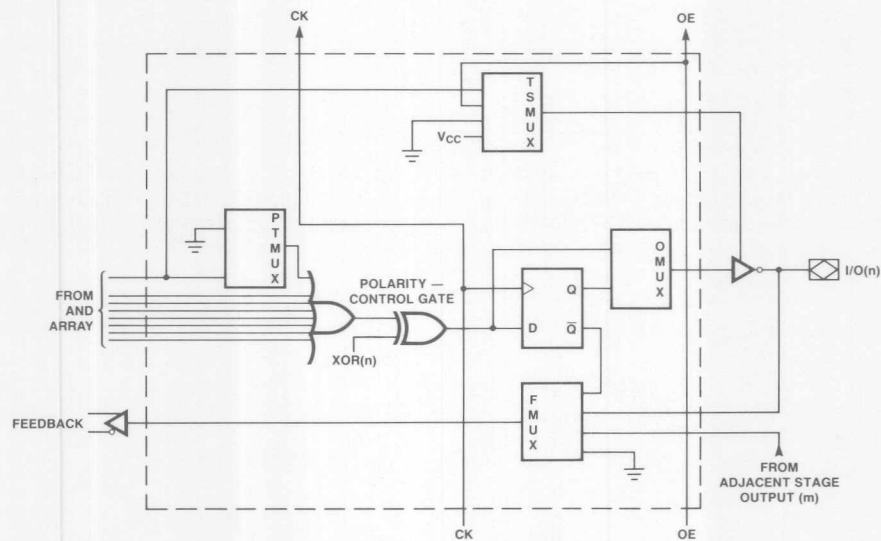
Data	Control	XOR Output
0	0	0
1	0	1
0	1	1
1	1	0

#### References:

- (1) A. Friedman, "Logical Design of Digital Systems," Computer Science Press Inc., CA, 1975.
- (2) H. Taub and D. Schilling, "Digital Integrated Electronics," McGraw-Hill, N.Y., 1977, Chapter 3.
- (3) E. Huntington, "Sets of Independent Postulates for the Algebra of Logic," Trans. American Mathematical Society, 1904, vol. 5, pp. 288-309.

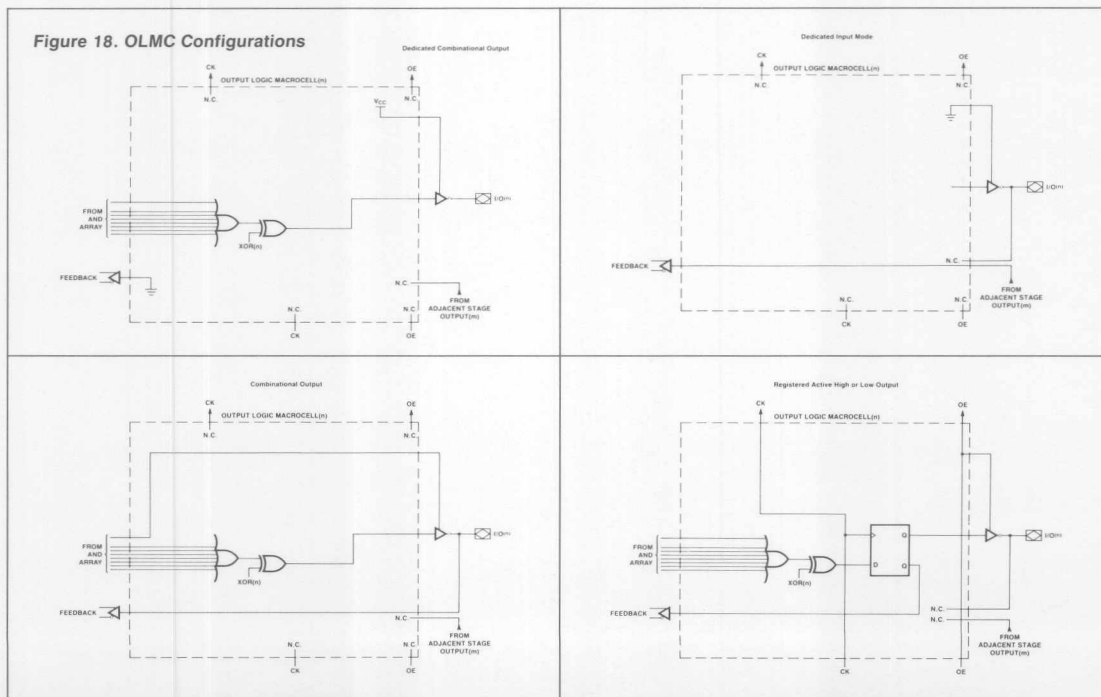


Figure 17. GAL Device Output Logic Macrocell (OLMC)



3

Figure 18. OLMC Configurations





<b>INTRODUCTION</b>	<b>1</b>
<b>GAL DEVICE SPECIFICATIONS</b>	<b>2</b>
<b>LOGIC TUTORIAL</b>	<b>3</b>
<b>USING DEVELOPMENT TOOLS</b>	<b>4</b>
<b>GAL DEVICE APPLICATIONS</b>	<b>5</b>
<b>TECHNICAL BRIEFS</b>	<b>6</b>
<b>E<sup>2</sup>CMOS TECHNOLOGY OVERVIEW</b>	<b>7</b>
<b>GAL DEVICE QUALITY AND RELIABILITY</b>	<b>8</b>
<b>ARTICLE REPRINTS</b>	<b>9</b>
<b>APPENDICES</b>	<b>10</b>
<b>SALES OFFICES</b>	<b>11</b>



## HARDWARE AND SOFTWARE TOOLS

Lattice Semiconductor specializes in the design and manufacture of high-speed E<sup>2</sup>CMOS programmable logic devices. It is not currently our intention to provide custom software and/or hardware for the purpose of developing patterns for the GAL family of devices, and, as such, we leave the software/hardware task to the respective experts in those fields.

Such third-party development-tool suppliers provide support for all major devices, from a variety of manufacturers. If you're just starting out with programmable logic and plan to purchase development tools, rest assured that industry-standard hardware and software will not only handle GAL device development, but can serve in those occasions where another manufacturer's PLD might be needed. If you're already using standard tools for PLD development, the move to GAL devices won't require sophisticated or expensive upgrades; current third-party development tools support Lattice GAL devices to their full extent. At most, an upgrade to the current revision of the support tool may be required.

Lattice's Applications Department remains on call to assist you in the task of development using third-party tools. Our engineers, trained on a variety of standard equipment, are prepared to answer any questions you may have. In addition, they are able to exploit the tools to more fully support the unique benefits of GAL devices. Here, we provide the basis for getting started with GAL devices. As you proceed with the development of your applications, call us — we'd like to hear how it's going.

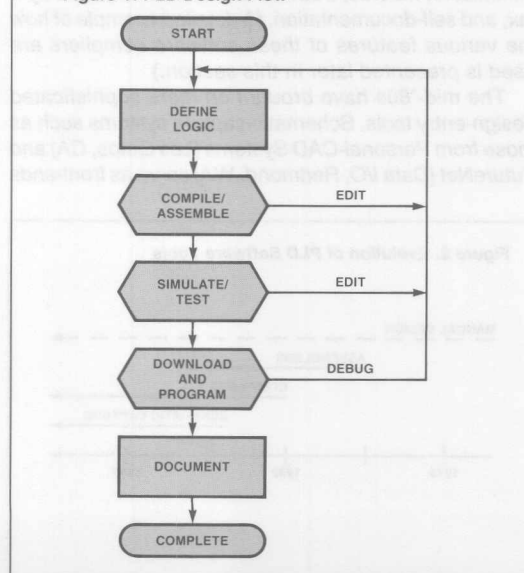
The typical PLD design flow, shown in Figure 1, begins with a design specification, iterates the logic to achieve proper functionality, and ends with a 'download' of the information to a programming fixture that patterns the device for the system. Critical to the accuracy and ultimate success of the PLD design process is the use of software-development tools to minimize the chance of error and improve design efficiency.

## Software Tools

Software tools have improved by orders of magnitude over the last decade (Figure 2). The early '70s was without development software. The designer was required to define the logic function in terms of 1,000 to 3,000 individual fuse locations — by hand. Each bit was painstakingly analyzed for its proper state, entered onto a logic map, and then typed into the programmer by hand. As expected, the process was fraught with errors, and ultimately inhibited the widespread acceptance of PLDs. The many types available today range from low-level assemblers to higher-level compilers; at the top

4

Figure 1. PLD Design Flow





end, sophisticated logic-extraction systems will 'capture' designers' gate-level schematics and implement the functions into PLDs.

Assembly-level programs appeared in the late '70s. The software was given away by the PLD manufacturers to support their respective devices. The most popular assembler — PALASM, from Monolithic Memories — was not without its shortcomings: unfriendly command structure, specific requirements for data structure and sequence, PAL device only, and no intelligence. Of course, the alternative — manual fuse pattern development — was far less palatable; of course, this software (and the PAL devices supported by it) enjoyed wide acceptance.

The limitations of the assembler became apparent when other companies and second-sources entered the PLD marketplace. MMI's software did not fully support competitive devices, and each subsequent manufacturer developed its own clumsy, limited software. The designer was forced to learn and support a half-dozen development packages. Fortunately, the age of compilers was about to dawn.

Compiler-based software was the response in the early '80s to the need for more flexible and advanced development tools. Two independent third-party companies — Assisted Technology (Los Gatos, CA) and a new division of Data I/O Corp. (Redmond, WA) entered the marketplace with their CUPL and ABEL packages, respectively, and simultaneously filled the need for generic software that supported all manufacturers, all device types, and had a common user interface.

The higher level of the compiler-based software supported such advanced design features as logic-equation minimizers, macros, truth-table and state-machine syntax, and self-documentation. (A detailed example of how the various features of these software compilers are used is presented later in this section.)

The mid-'80s have brought on more sophisticated design-entry tools. Schematic-capture systems such as those from Personal-CAD Systems (Los Gatos, CA) and FutureNet (Data I/O, Redmond, WA) serve as front-ends

to the design-entry process, allowing a graphic representation of a logic schematic as the input format for a translator, as shown in Figure 3. The translator converts the graphics representation to a network list that is then compiled by a PLD software tool.

The schematic-capture entry format is ideal both to upgrade existing designs (whose logic diagrams have already been drawn) for incorporating into PLDs, and to initiate and thereby automatically document new designs with logic diagrams, PC-board layouts, and PLD documentation files.

### Hardware Tools

The same arguments as those expressed above for universal software tools apply to universal hardware. Hardware that is developed by third parties is more flexible and provides a future growth path to the user. Lattice recommends the use of third-party programming hardware for GAL device development.

Universal programming hardware allows the programming of a variety of devices without the aid of custom fixtures or manufacturers' adapters. Since the Lattice GAL programming algorithm requires no abnormal voltages or timings, as some one-time programmable technologies do, most all hardware manufacturers support GAL devices on existing models.

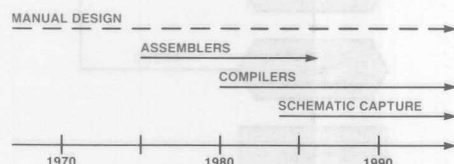
Patterning the PLD is the process of providing it with the data (the JEDEC file) to perform a specific custom function, and applying the appropriate series of voltage pulses. 'Support' by a hardware manufacturer refers to his ability to provide the appropriate voltage pulses and timings for a given PLD. After that, patterning a device merely requires downloading the JEDEC file.

Downloading is the process of 'teaching' the hardware programmer the pattern that is necessary to program a device. This data can come from a pre-patterned device (or 'master'), from a computer (via direct connection or modem) or from an attached peripheral, such as a tape drive. If the file is transferred in JEDEC format, as most are, a checksum is calculated and verified at the end of the data transfer to ensure that no data was dropped or garbled during transmission. Most programmers have either a single button or simple command string that puts the hardware into the download mode.

The programming of the GAL device is controlled by the programming hardware. Since the GAL device uses a nonvolatile, reprogrammable E<sup>2</sup>CMOS technology, the device can be erased; in fact, the device is automatically erased as the first step in the programming algorithm.

The patterning of the GAL device array is done using a parallel-programming scheme, which keeps the total programming time to well under a second. The algorithm is so efficient that it programs devices nearly 50% faster than typical bipolar PLD algorithms, and an order

Figure 2. Evolution of PLD Software Tools



of magnitude faster than UV-CMOS approaches. During this programming time, both the logic array and the architecture matrix are patterned.

Finally, an analog verify of each and every cell in the GAL device takes place, to ensure that the cell is fully programmed and will retain data for a minimum of 20 years.

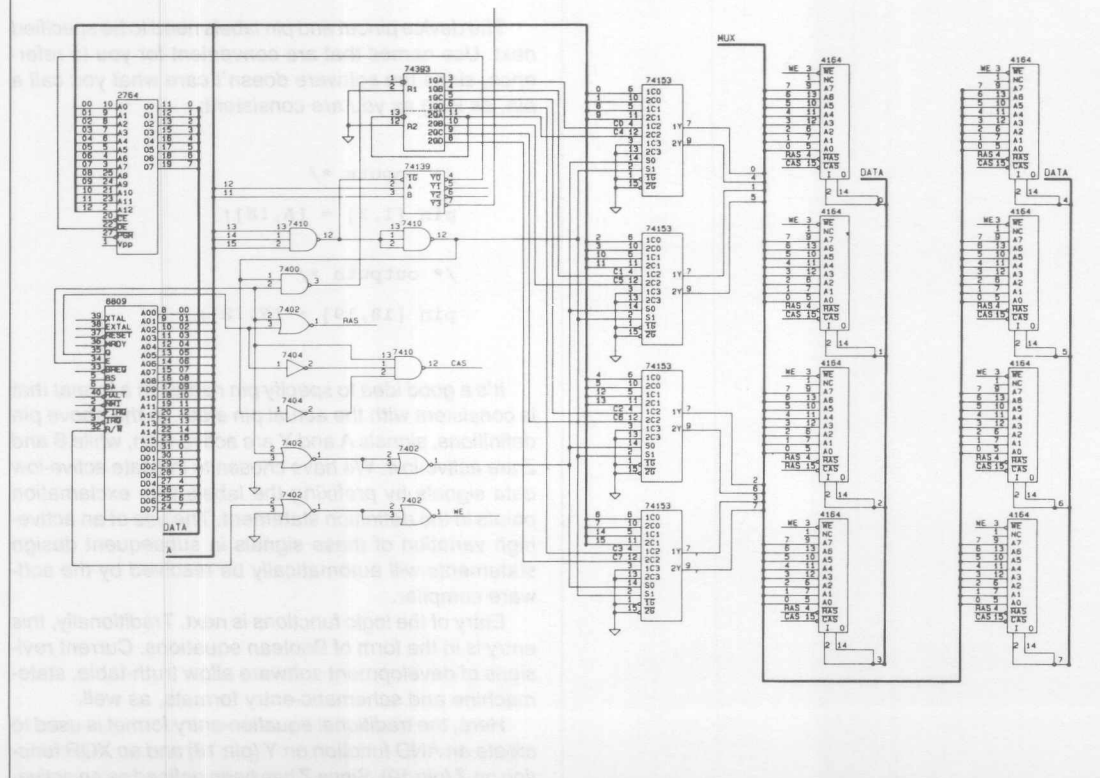
It is worth noting here that GAL devices offer a security cell that can be programmed to prevent examination (or further verification) of the pattern in the programmable arrays — a feature provided so that a proprietary design can be obscured from competitive or enemy eyes.

Somewhat ironically, the GAL device security cell is itself erasable; it can only be erased, however, in conjunction with an array 'Bulk Erase,' during which all bits are cleared at once. This allows the designer or manufacturing person to reuse previously secured devices — a feature never before available in PLDs.

## Debugging and Pattern Revisions

GAL devices bring extensive advantages to the manufacturing and design engineering areas, due to their unique combination of E<sup>2</sup>CMOS technology, generic architecture, and unmatched quality levels. Only GAL devices are instantly erasable in a standard hardware programming fixture. As mentioned, erasure takes place automatically just prior to the re-patterning of the array. No time-consuming 'trips to a UV lamp' are necessary, as with UV-erasable PLDs. Both the GAL device's logic array and device architecture configuration are fully reprogrammable and reconfigurable. In addition, the erasable GAL device is assembled in a low-cost plastic package, not an expensive quartz-windowed package. Pattern revisions can be recorded in the device's electronic signature, allow the traceability, tracking and verification of every device. Finally, inventories are kept to a minimum, thanks to the generic 'one device fits all' macrocell approach. □

Figure 3. Example of Input for a Schematic-Capture Based System



## THE DESIGN PROCESS

By choosing generic, compiler-based software, generic hardware and generic silicon (such as GAL devices), the biggest decisions in the design process have already been made. The choice of the appropriate programmable logic device has traditionally been a difficult first step in starting a design, since with bipolar PLDs, you must guess which one of the dozens of architectures has the right combination of outputs, I/Os and registers. If your choice is wrong, you must guess again. The Lattice GAL concept simplifies the approach, requiring that you merely count the number of inputs and outputs, then select a speed/power option. The development software automatically and dynamically allocates the inputs, I/Os, registers, and so on.

The following design example excerpts material from the first application, 'Basic Gates,' in Section 5 of this handbook. The specific syntax is that of CUPL; however, other generic software (ABEL) has similar syntax and functions. In this cursory 'walk-through', segments of code are presented as they would appear on the screen of the personal computer running the CUPL software. The manufacturers of the software would, of course, be glad to provide a more comprehensive tutorial.

Once the software knows which device will be used, fields are provided for optional information, such as company name, design description, etc.:

```

/*****
/*
/*      Tutorial Using a GAL16V8
/*
/*      Source File (401.PLD)
/*
*****/

```

```

PARTNO 99;
NAME CHAP4;
REV 1;
DATE 4/16/86;
DESIGNER Dean Suhr;
COMPANY Lattice Semiconductor;
ASSEMBLY n/a;
LOCATION n/a;

```

The device pinout and pin labels need to be specified next. Use names that are convenient for you to reference, since the software doesn't care what you call a pin, as long as you are consistent:

```

/* inputs */

pin [1,2] = [A,!B];

/* outputs */

pin [18,19] = [Y,!Z];

```

It's a good idea to specify pin names in a format that is consistent with the actual pin state. In the above pin definitions, signals A and Y are active-high, while B and Z are active-low. We have chosen to indicate active-low data signals by prefixing the labels with exclamation points in the definition statement. The use of an active-high variation of these signals in subsequent design statements will automatically be resolved by the software compiler.

Entry of the logic functions is next. Traditionally, this entry is in the form of Boolean equations. Current revisions of development software allow truth-table, state-machine and schematic-entry formats, as well.

Here, the traditional equation-entry format is used to create an AND function on Y (pin 18) and an XOR function on Z (pin 19). Since Z has been defined as an active-

low signal, however, we will actually end up with XNOR on pin 19:

```
/* logic equations */
Y = A & B;
Z = A & B # !A & !B;
```

The operators used in the CUPL language are '!' for invert, '&' for the AND function and '#' for the OR function. The equations are written exactly as needed. All of the inversions for active-low inputs and outputs will be automatically resolved, a routine procedure for compiler software. Although these are simple equations, had they been complex ones that needed automatic reduction to a specific number of product terms for a given PLD, the software would have performed the reduction, as well.

Next, the CUPL compiler needs to be invoked to process the 'source' file, the text and equations provided above. For the compiler to run one of its 'modules,' it must be told the target device type ('G16V8' in the example), source file name ('401'), and whatever additional functions the user would have it perform, through the use of flags ('-jlfxs') that are passed at compile time. The compile directive would typically appear as follows:

```
CUPL -jlfxs G16V8 401

CUPL Version 2.10B1 Copyright (c)
1983,84,85 Assisted Technology,
Inc.

cuplx
time: 3 secs

cupla
time: 19 secs

cuplb
time: 7 secs

cuplm
time: 4 secs

cuplc
time: 15 secs

csima
time: 24 secs

total time: 73 secs
```

Some of the more recently announced PLD support products such as the Personal Silicon Foundry software available from Data I/O use a very friendly menu concept that saves defaults, eliminating the need to use flags.

The CUPL compiler produces a report, called a documentation file, part of which is presented here for reference. The purpose of the file is to provide a hard-copy documentation of the final (reduced) equations, the cell map or 'fuse plot,' and a chip-pinout diagram, if desired:

```
Device G16V8s Library DLIB-d-55-8
Created Wed Apr 16 04:52:24 1986
```

#### Expanded Product Terms

```
Y => A & B
Z => A & B # !A & !B
```

#### Symbol Table

Pin	Variable	Pin	Pterms	Max
Pol	Name		Used	Pterms
	A	1	-	-
!	B	2	-	-
	Y	18	1	8
!	Z	19	2	8

#### Fuse Plot

```
Syn 2192 - Ac0 2193 x
Pin #19 2048 Pol x 2120 Ac1 x

0000 -xx-----
0032 x--x-----
0064 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0096 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0128 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0160 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0192 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0224 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Pin #18 2049 Pol - 2121 Ac1 x
0256 -xx-----
0288 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0320 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0352 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0384 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0416 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0448 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0480 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

(continued ...)

```
LEGEND X : fuse not blown
        - : fuse blown
```



```
*QP20
*GF2194
*G0
*F0
*L0000 10011111111111111111111111111111
*L0032 01101111111111111111111111111111
*L0256 10011111111111111111111111111111
*L2048 01000000000000000000000000000000
*L2112 00000000000111111111111111111111
*L2144 11111111111111111111111111111111
*L2176 11111111111111111111111111111111
*C14D6
```



## EXAMPLE: A TWO-STORY ELEVATOR

This example is designed to step the reader through the process of creating and implementing a logic design using GAL devices. Whether a novice or intermediate user of PLDs, the reader is encouraged to familiarize himself with this section, as well as with the applications in Section 5, which provides examples of how to implement basic functions such as decoders, shifters, multiplexers, counters, and so on.

Here, we will be building a two-story elevator control unit. The function of the unit is to monitor the state of the call buttons, respond to calls for service, and display the status of the elevator by means of floor and direction displays. The operating control function requires a small state machine and a latch function, while the display logic uses only combinational circuits.

Our elevator (Figure 1) travels between two floors. Arriving at a floor in response to a call for service, the elevator opens its doors, pauses, then closes them automatically. If the Up or Down button is pushed, the elevator travels to the other floor. A microswitch mounted on the car informs the controller that the elevator has arrived at a new floor.

Once the elevator arrives at a floor to discharge passengers, it opens its doors, pauses to let the passengers out, then closes the doors and assumes its wait position. A call for service at the floor where the elevator is resting will result in the doors being opened.

A free-running clock controls the elevator's operation, toggling every 5 to 10 seconds to allow a brief pause during each arrival and departure activity. While this slow clock rate is appropriate for the timing of the elevator doors and car movement, it is far too slow to capture a time-independent call for service. As such, a latch function that captures data instantly (actually within 25 ns) is designed using two of the GAL device macrocells.

As shown in Figure 2, the total elevator control unit uses two GAL16V8s — one to perform the actual control function, the other to handle the display.

The control of the elevator comprises two basic functions: the call-button latches and the state machine. The latches, constructed from the GAL device's available

Figure 1. Elevator Scenario

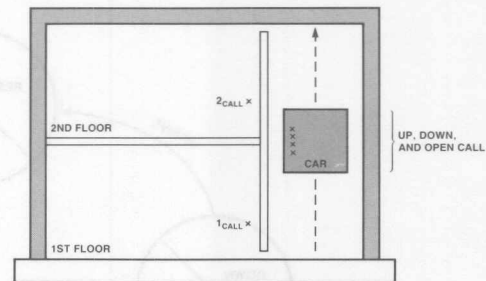
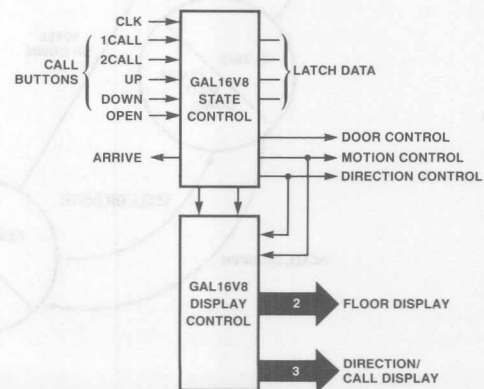


Figure 2. Block Diagram



AND and OR gates (instead of using the on-chip D-type register), are instantaneous and not dependent on a clock for holding data. The logic diagram of the S-R latch used is shown in Figure 3. As shown in the accompanying truth table, the latch is set by applying a logic '1' to SET, and reset by applying a logic 1 to RESET. Applying a logic 0 to both inputs causes a hold state, while applying a logic 1 to both is undefined for this type of latch. The various call signals — UCALL, DCALL, OCALL, 1CALL, 2CALL — are applied to the two latches to command the elevator to travel to the requested floor.

The first step in this GAL implementation is to translate the functional operation of the elevator (described in the text in the preceding paragraphs) to a logical format. This is realized through the use of a state-transition diagram, which literally describes all the allowed stable

Figure 3. SR Latch

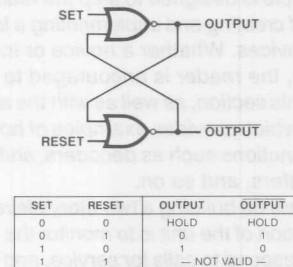
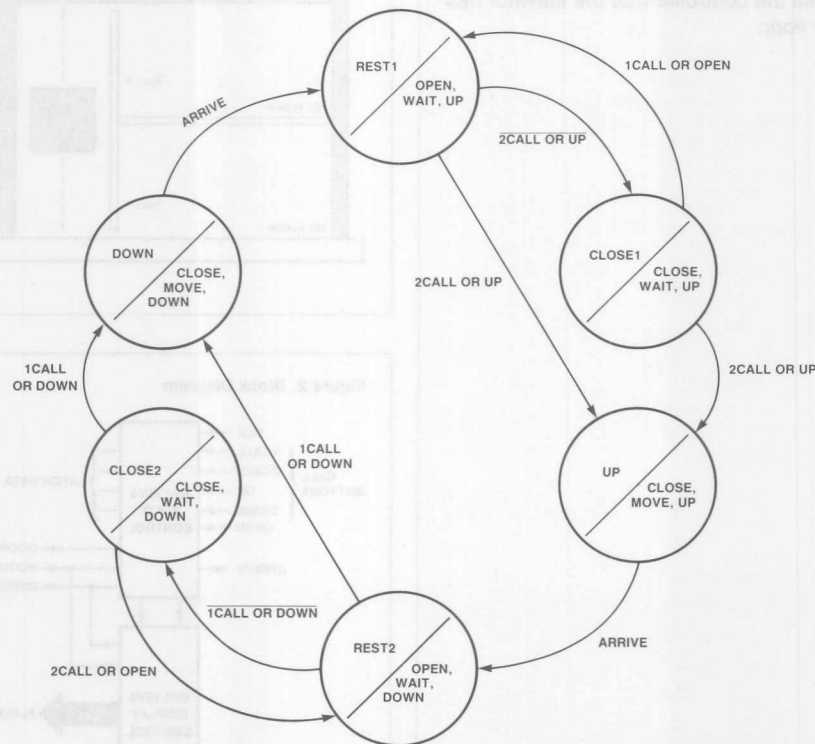


Figure 4. State-Transition Diagram



states (on floor two, doors open, etc.) that our elevator can be in. An unacceptable state, for example, would be resting between floors.

Figure 4 shows the state-transition diagram. Inside each state circle, the diagram indicates the state name (top half) and the condition of each of the state variables: DOOR, MOTION, and DIRECTION. The transitions out of the state are shown with the logic level requirements to make the transition. Also shown is the destination of each of the transfers.

The latched signals L1CALL and L2CALL are used to start the states changing. The ARRIVAL input tells the car when to stop its motion. The normal wait state of the elevator is either CLOSE1 or CLOSE2 (not moving with door closed).

The information is then transferred from the transition diagram to the CUPL state-machine syntax, shown in Figure 5. Notice the use of defaults in the state syntax to indicate what state should be selected (or held) if none of the criteria for exit is met. There is also an identi-

Figure 5. Design Input File for Control Section

```

/*****
/*
/* Two Story Elevator Example using a GAL16V8
/*
/* Control Logic
/*
/* CUPL Source File (Elev_ctl.PLD)
*****/

PARTNO 01;
NAME Elev_ctl;
REV 0;
DATE 4/17/86;
DESIGNER Dean Suhr;
COMPANY Lattice Semiconductor;
ASSEMBLY Control Board;
LOCATION n/a;

/* device */

device gl6v8;

/* inputs */

pin [1,11] = [CLK,!OE];
pin [2,3] = [lCall,2Call]; /* Buttons on Floors */
/* Up Down Open
pin [4,5,6] = [uCall,dCall,oCall]; /* Buttons in elevator
pin [7] = [Arrive]; /* Floor arrival sensor

fld Control = [Door, Motion, Direction];

/* outputs */

pin [12] = [Door]; /* 0 = Open, 1 = Close */
pin [13] = [Motion]; /* 0 = Wait, 1 = Move */
pin [14] = [Direction]; /* 0 = Up, 1 = Down */
/* Floor : 0 = One, 2 = Two

pin [16,17] = [L1Call,L1Call_bar]; /* Latched call to 1st floor */
pin [18,19] = [L2Call,L2Call_bar]; /* Latched call to 2nd floor */

```

fiable 1-to-1 correspondence from the state transition diagram to the state-machine syntax. The documentation file, which includes reduced equations, expanded product terms, symbol table, 'fuse plot', JEDEC file, and pinout diagram, is shown in Figures 6 through 10. Notice that the compiler automatically chose the proper polarity to fit the reduced equations into the GAL device, using DeMorgan's Law: pins 12, 13, and 14 are inverted, relative to the other output pins.

## Display Design

The source file for the up/down arrow display is shown in Figure 11. The UPARROW is active only when the car is moving up. DNARROW is true only when the car is moving down. The common bar, SEGARROW, is active during any call, in any direction. This signal is also active

Figure 5. (cont'd)

```

/* logic equations */
1Floor      = Direction;
2Floor      = !Direction;

L1Call      = !( L1Call_bar # 1Call # (1Floor & oCall)
                # (2Floor & dCall) );

L1Call_bar  = !( L1Call          # (!Door & !Motion & !Direction));

L2Call      = !( L2Call_bar # 2Call # (2Floor & oCall)
                # (1Floor & uCall) );

L2Call_bar  = !( L2Call          # (!Door & !Motion & Direction));

/* State Definitions */

/* Door Motion Direction */
$define Rest1      'b'000
$define Close1     'b'100
$define Up         'b'110
$define Rest2      'b'001
$define Close2     'b'101
$define Down       'b'111

sequence Control {

Present Rest1:
    If ( L2Call ) next Up;
    default next Close1;

Present Close1:
    If ( L2Call ) next Up;
    If ( L1Call ) next Rest1;
    default next Close1;

```

when an unserviced call is active. As such, the SEG-ARROW signal is a call waiting indicator that acknowledges a call button being pushed. The logic equations for the arrow functions are self-explanatory; its input signals come from the controller.

The floor indicator is a simplified decoder. A truth-table input format is used for the design. Notice that a

floor is always indicated, and that the change occurs when the direction bit changes. This bit is constrained by the state machine to change only when the car arrives at a floor. The documentation file showing the reduced equations, expanded product terms, symbol table, 'fuse plot', JEDEC file, and pinout diagram is reproduced in Figures 12 through 16. □

Figure 5. (cont'd)

```

Present Up:
  If (Arrive) next Rest2;
  default next Up;

Present Rest2:
  If ( L1Call ) next Down;
  default next Close2;

Present Close2:
  If ( L1Call ) next Down;
  If ( L2Call ) next Rest2;
  default next Close2;

Present Down:
  If (Arrive) next Rest1;
  default next Down;
}

```



Figure 6. Expanded Product Terms for Control Section

```

=====
Expanded Product Terms
=====

1Floor =>
    Direction

2Floor =>
    !Direction

Control =>
    Door , Motion , Direction

Direction.d =>
    !Arrive & Direction & Door & Motion
    # Arrive & !Direction & Door & Motion
    # Direction & !Door & !Motion
    # Direction & Door & L1Call & !Motion
    # Direction & Door & L2Call & !Motion
    # Direction & Door & !L1Call & !L2Call & !Motion

Door.d =>
    Direction & Door & L1Call & !Motion
    # !Direction & Door & L2Call & !Motion
    # !Door & !Motion
    # Door & !L1Call & !L2Call & !Motion
    # !Arrive & Door & Motion

L1Call =>
    L1Call_bar
    # 1Call
    # Direction & oCall
    # !Direction & dCall

L1Call_bar =>
    L1Call
    # !Direction & !Door & !Motion

L2Call =>
    L2Call_bar
    # 2Call
    # !Direction & oCall
    # Direction & uCall

L2Call_bar =>
    L2Call
    # Direction & !Door & !Motion

Motion.d =>
    !Direction & L2Call & !Motion
    # Direction & L1Call & !Motion
    # !Arrive & Door & Motion

```

Figure 6. (cont'd)

```

L1Call.oe =>
1

L1Call_bar.oe =>
1

L2Call.oe =>
1

L2Call_bar.oe =>
1

```

Figure 7. Symbol Table for Control Section

Symbol Table							
Pin Pol	Variable Name	Ext	Pin	Type	Pterms Used	Max Pterms	Min Level
	1Call		2	V	-	-	-
	1Floor		0	I	1	-	-
	2Call		3	V	-	-	-
	2Floor		0	I	1	-	-
	Arrive		7	V	-	-	-
	CLK		1	V	-	-	-
	Control		0	F	-	-	-
	Direction		14	V	-	-	-
	Direction	d	14	X	6	8	1
	Door		12	V	-	-	-
	Door	d	12	X	5	8	1
	L1Call		16	V	4	7	1
	L1Call_bar		17	V	2	7	1
	L2Call		18	V	4	7	1
	L2Call_bar		19	V	2	7	1
	Motion		13	V	-	-	-
	Motion	d	13	X	3	8	1
!	OE		11	V	-	-	-
	dCall		5	V	-	-	-
	oCall		6	V	-	-	-
	uCall		4	V	-	-	-
	L1Call	oe	16	D	1	1	0
	L1Call_bar	oe	17	D	1	1	0
	L2Call	oe	18	D	1	1	0
	L2Call_bar	oe	19	D	1	1	0

LEGEND      F : field      D : default variable      M : extended node  
               N : node      I : intermediate variable      T : function  
               V : variable      X : extended variable      U : undefined

Figure 8. 'Fuse' Plot for Control Section

```

=====
                          Fuse Plot
=====

Syn      2192 x Ac0      2193 -

Pin #19  2048 Pol x  2120 Ac1 -
0000 -----
0032 -----x-----
0064 -----x-----x---x---x
0096 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0128 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0160 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0192 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0224 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Pin #18  2049 Pol x  2121 Ac1 -
0256 -----
0288 -----x-----
0320 -----x-----
0352 -----x-----x-----
0384 -----x-----x-----
0416 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0448 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0480 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Pin #17  2050 Pol x  2122 Ac1 -
0512 -----
0544 -----x-----
0576 -----x-----x---x---x
0608 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0640 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0672 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0704 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0736 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Pin #16  2051 Pol x  2123 Ac1 -
0768 -----
0800 -----x-----
0832 x-----
0864 -----x-----x-----
0896 -----x-----x-----
0928 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0960 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0992 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Pin #15  2052 Pol x  2124 Ac1 -
1024 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1056 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1088 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1120 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1152 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1184 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1216 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1248 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Pin #14  2053 Pol -  2125 Ac1 x
1280 -----xx---x---x---x---
1312 -----x---x---x---x---
1344 -----x---x---x---x---
1376 -----x---x---x---x---
1408 -----x---x---x---x---
1440 -----x---x---x---x---
1472 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1504 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Pin #13  2054 Pol -  2126 Ac1 x
1536 -----x---x---x---x---
1568 -----x---x---x---x---
1600 -----x---x---x---x---
1632 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1664 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1696 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1728 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1760 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Pin #12  2055 Pol -  2127 Ac1 x
1792 -----x---x---x---x---
1824 -----x---x---x---x---
1856 -----x---x---x---x---
1888 -----x---x---x---x---
1920 -----x---x---x---x---
1952 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1984 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
2016 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

```

LEGEND: X: Programmed Cell  
-: Erased Cell

Figure 9. JEDEC File for Control Section

```

*QP20
*QF2194
*G0
*F0
*L0000 11111111111111111111111111111111
*L0032 11111101111111111111111111111111
*L0064 11111111111111111111110111101110
*L0256 11111111111111111111111111111111
*L0288 11011111111111111111111111111111
*L0320 11110111111111111111111111111111
*L0352 11111111111111110111110111111111
*L0384 11111111011111111111111101111111
*L0512 11111111111111111111111111111111
*L0544 11111111111111011111111111111111
*L0576 1111111111111111111111011101110
*L0768 11111111111111111111111111111111
*L0800 11111111101111111111111111111111
*L0832 01111111111111111111111111111111
*L0864 11111111111111111111110111111111
*L0896 11111111111101111111111011111111
*L1280 1111111111111111111111001110101101
*L1312 111111111111111111111101101101101
*L1344 111111111111111111111110111101110
*L1376 11111111111111011111110111101101
*L1408 11111101111111111111110111101101
*L1440 111111101111111101111110111101101
*L1536 111111011111111111111111011101111
*L1568 11111111111111011111110111101111
*L1600 11111111111111111111110111101101
*L1792 11111111111111111111110111101101
*L1824 111111011111111111111110111101101
*L1856 111111111111111111111111101110
*L1888 11111110111111110111111111101101
*L1920 111111111111111111111101111101101
*L2048 00000111000000000000000000000000
*L2112 00000000111110001111111111111111
*L2144 11111111111111111111111111111111
*L2176 111111111111111101
*C7175
*51DE

```

Figure 10. Pinout Diagram for Control Section

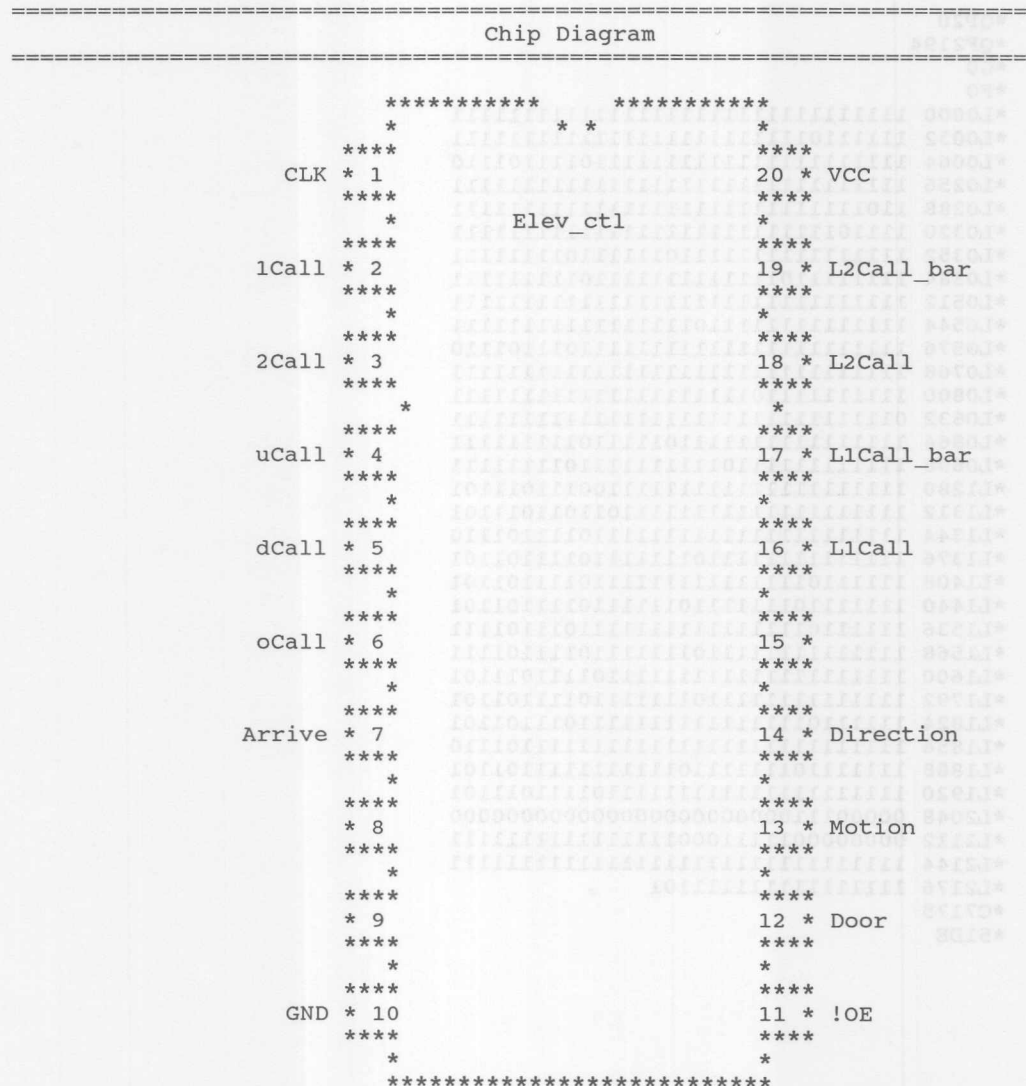






Figure 11. (cont'd)

```

/* logic equations */

UpArrow = Motion & Direction ;
SegArrow = L1Call # L2Call ;
DnArrow = Motion & !Direction ;

/*
TABLE INPUTS      TABLE OUTPUTS
*/
table      Direction, Motion => Seg1, Seg2, Seg12 {
    'b'00      =>      'b'101;
    'b'01      =>      'b'101;
    'b'10      =>      'b'011;
    'b'11      =>      'b'011; }

```

Figure 12. Expanded Product Terms for Display Section

```

=====
Expanded Product Terms
=====

DnArrow =>
    !Direction & Motion

Seg1 =>
    Direction

Seg12 =>
    0

Seg2 =>
    !Direction

SegArrow =>
    L1Call
    # L2Call

UpArrow =>
    Direction & Motion

```

Figure 13. Symbol Table for Display Section

Symbol Table							
Pin	Variable						
Pol	Name	Ext	Pin	Type	Pterms	Max	Min
---	----	---	---	----	-----	Pterms	Level
	Direction		5	V	-	-	-
	DnArrow		14	V	1	8	1
	L1Call		2	V	-	-	-
	L2Call		3	V	-	-	-
	Motion		4	V	-	-	-
	Seg1		15	V	1	8	1
	Seg12		17	V	1	8	1
	Seg2		16	V	1	8	1
	SegArrow		13	V	2	8	1
	UpArrow		12	V	1	8	1
LEGEND							
F	: field	D	: default variable	M	: extended node		
N	: node	I	: intermediate variable	T	: function		
V	: variable	X	: extended variable	U	: undefined		

Figure 14. 'Fuse' Plot for Display Section

```

=====
                        Fuse Plot
=====

Syn 2192 - Ac0 2193 x

Pin #19 2048 Pol x 2120 Ac1 -
0000 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0032 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0064 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0096 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0128 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0160 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0192 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0224 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #18 2049 Pol x 2121 Ac1 -
0256 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0288 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0320 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0352 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0384 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0416 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0448 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0480 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #17 2050 Pol - 2122 Ac1 x
0512 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0544 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0576 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0608 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0640 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0672 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0704 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0736 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #16 2051 Pol - 2123 Ac1 x
0768 -----x-----
0800 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0832 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0864 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0896 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0928 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0960 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0992 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Pin #15 2052 Pol - 2124 Ac1 x
1024 -----x-----
1056 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1088 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1120 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1152 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1184 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1216 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1248 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #14 2053 Pol - 2125 Ac1 x
1280 -----x-----
1312 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1344 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1376 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1408 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1440 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1472 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1504 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #13 2054 Pol - 2126 Ac1 x
1536 x-----
1568 ----x-----
1600 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1632 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1664 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1696 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1728 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1760 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #12 2055 Pol - 2127 Ac1 x
1792 -----x-----
1824 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1856 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1888 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1920 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1952 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1984 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
2016 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

```

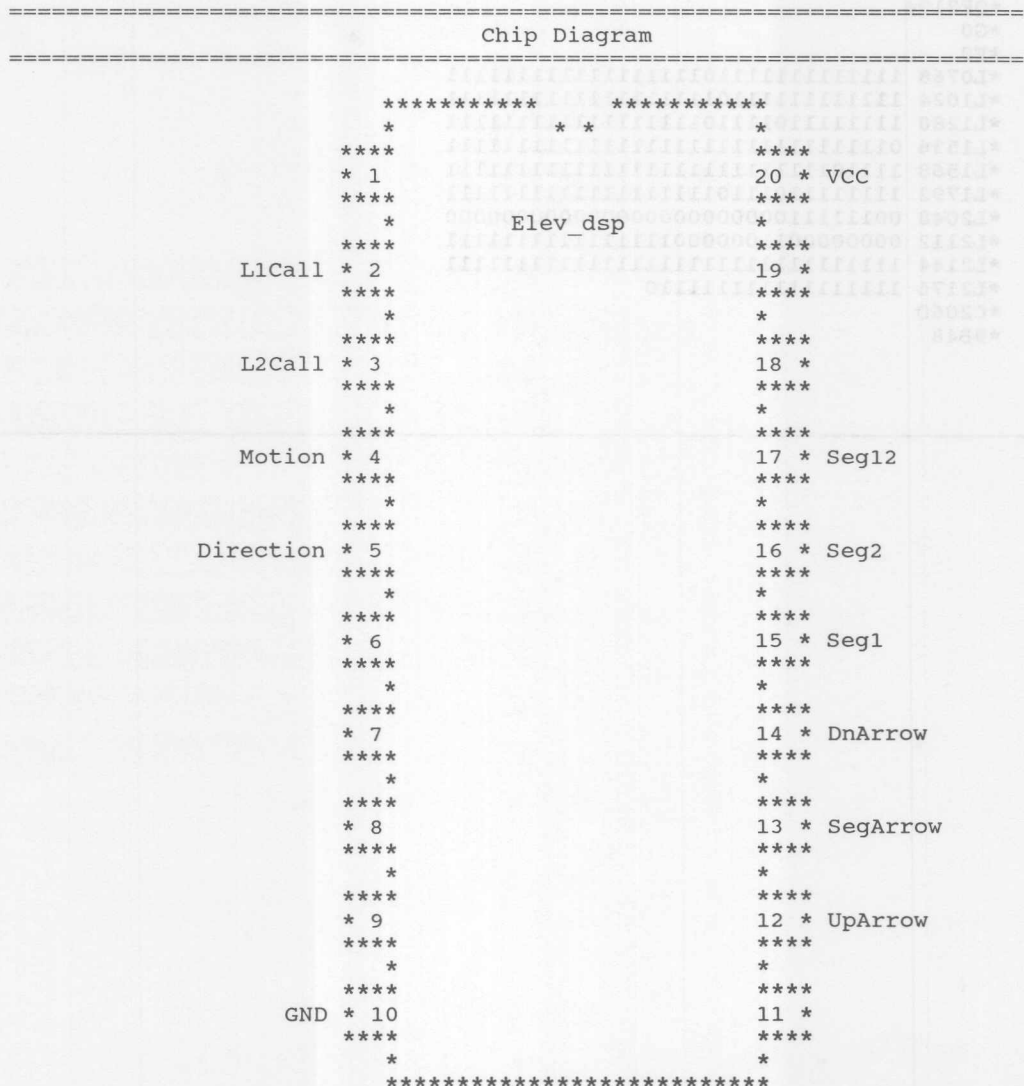
LEGEND: X: Programmed Cell  
 -: Erased Cell

Figure 15. JEDEC File for Display Section

```
*QP20
*QF2194
*G0
*F0
*L0768 111111111110111111111111111111
*L1024 111111111110111111111111111111
*L1280 111111110111101111111111111111
*L1536 011111111111111111111111111111
*L1568 111101111111111111111111111111
*L1792 111111101110111111111111111111
*L2048 001111110000000000000000000000
*L2112 000000001100000001111111111111
*L2144 111111111111111111111111111111
*L2176 11111111111111110
*C206D
*9B48
```



Figure 16. Pinout Diagram for Display Section



## Introduction to GAL Applications

are discussed, ranging from simple logic blocks to complex interfacing and control functions. Each application example is presented as a GAL design, comparing with the standard logic function and the actual GAL device logic implementation. The GAL design specification and the actual CUP or ABEL output for the program are also included. The examples are intended to serve as guides for designers using GAL devices in their own systems. These examples, combined with the GAL design information in the rest of the book, will help designers become comfortable with GAL design.

The Lattice Semiconductor GAL family brings new degrees of freedom to the field of logic design. The GAL device is an extremely versatile new PLD whose applications are practically unlimited. Besides the benefit of its many applications, the designer can use GAL devices to replace existing conventional programmable logic and to optimize the design of new products. Throughout previous sections, the designer has become familiar with the GAL concept and with design techniques and advantages gained when using GAL devices. In this section, a few of the many applications

<b>INTRODUCTION</b>	<b>1</b>
<b>GAL DEVICE SPECIFICATIONS</b>	<b>2</b>
<b>LOGIC TUTORIAL</b>	<b>3</b>
<b>USING DEVELOPMENT TOOLS</b>	<b>4</b>
<b>GAL DEVICE APPLICATIONS</b>	<b>5</b>
<b>TECHNICAL BRIEFS</b>	<b>6</b>
<b>E<sup>2</sup>C MOS TECHNOLOGY OVERVIEW</b>	<b>7</b>
<b>GAL DEVICE QUALITY AND RELIABILITY</b>	<b>8</b>
<b>ARTICLE REPRINTS</b>	<b>9</b>
<b>APPENDICES</b>	<b>10</b>
<b>SALES OFFICES</b>	<b>11</b>

## Introduction to GAL Applications

The Lattice Semiconductor GAL family brings new degrees of freedom to the field of logic design. The GAL device is an extremely versatile new PLD whose applications are practically unlimited. Besides the benefit of its many applications, the designer can use GAL devices to replace existing conventional programmable logic and to optimize the design of new products.

Throughout previous sections, the designer has become familiar with the GAL concept, and with design techniques and advantages gained when using GAL devices. In this section, a few of the many applications

are discussed, ranging from simple logic-gate replacement to complex interfacing and control functions. Each applications example is presented as a GAL design, complete with the required logic function and the actual GAL device logic implementation. The GAL design specification and the actual CUPL or ABEL output for the programmer make the examples complete enough to serve as guides for designers using GAL devices in their own systems. These examples, combined with the GAL design information in the rest of the book, will help designers become comfortable with GAL design. □

## DESCRIPTION

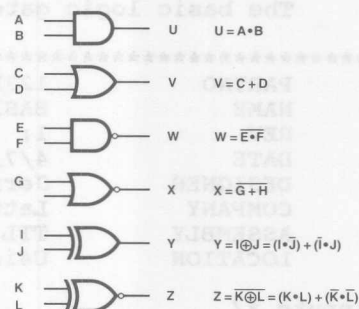
As a beginning example, the implementation of the basic gates: AND, OR, NAND, NOR, XOR, and XNOR (Figure 1) in a GAL16V8 is shown. The NOT function was omitted because its implementation is straightforward. Since 12 inputs and 6 outputs are needed (Figure 2), 2 Output Logic Macrocells (OLMCs) must be configured as dedicated inputs and 6 as dedicated combinational outputs. Programming software automatically handles this task. Active-high or -low outputs are no problem either, because of the programmable polarity feature of the GAL16V8.

Because the program has been compiled using CUPL, two files are needed: the logic design input file and the simulation file. For the design input file (Figure 3), an equation must be written for each output, and pin assignments must be made. The simulation file (Figure 4) contains 'vectors' that test the functionality of the device. GAL devices are completely tested at the factory and post-programming yields are guaranteed to be 100%; therefore, device testing is unnecessary. However, since test vectors are useful for design verification, they are included here.

The JEDEC file for this particular application is shown in Figure 5. Because of the GAL device's fuse-map compatibility with PAL devices, JEDEC files can be copied directly from PAL device applications into GAL devices.

As shown in Figure 6, CUPL software will produce pinout diagrams for your applications. The CUPL 'fuse' plot is shown in Figure 7; note the correspondence between it and the 'fuse' locations of the GAL16V8 logic diagram (Figure 8). □

Figure 1. Basic Logic Gates



5

Figure 2. GAL16V8 Basic Gates Pinout

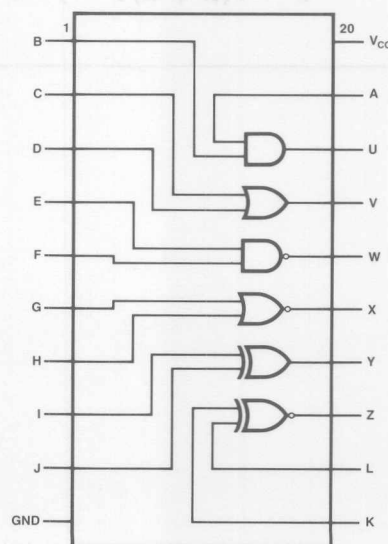


Figure 3. CUPL Logic Design Input File

```

/*****
/*
/*          CUPL INPUT FILE          */
/*      The basic logic gates implemented in a GAL16V8      */
/*
/*****

PARTNO      123XYZ;
NAME        BASICGAT;
REV         1;
DATE        4/7/86;
DESIGNER     Jerry;
COMPANY      Lattice Semiconductor;
ASSEMBLY     TTL Replacement;
LOCATION      U4ia;

/* inputs */
pin [19,1,2,3,4,5,6,7,8,9,11,12] = [A,B,C,D,E,F,G,H,I,J,K,L];

/* outputs */

pin [18,17,16,15,14,13] = [U,V,W,X,Y,Z];

/* logic equations */

U = A & B;
V = C # D;
W = !(E & F);
X = !(G # H);
Y = I $ J;      /* Y = (I & !J) # (!I & J) */
Z = !(K $ L);   /* Z = (K & L) # (!K & !L) */

```



Figure 4. CUPL Simulation File

```

/*****
/*
/*          CUPL SIMULATION FILE
/*      The basic logic gates implemented in a GAL16V8
/*
/*
/*****
PARTNO      123XYZ;
NAME        BASICGAT;
REV         1;
DATE        4/7/86;
DESIGNER    Jerry;
COMPANY     Lattice Semiconductor;
ASSEMBLY    TTL Replacement;
LOCATION     U4ia;

/* The Order Statement specifies the layout of the vector table
   in the simulation output file. %n = n spaces inserted between
   variables. */

order:  A,B,%1,U,%2,C,D,%1,V,%2,E,F,%1,W,%2,
        G,H,%1,X,%2,I,J,%1,Y,%2,K,L,%1,Z;

vectors:

/* AB U  CD V  EF W  GH X  IJ Y  KL Z */

00 L  XX X  XX X  XX X  XX X  XX X  /* test AND gate */
01 L  XX X  XX X  XX X  XX X  XX X
11 H  XX X  XX X  XX X  XX X  XX X
XX X  00 L  XX X  XX X  XX X  XX X  /* test OR gate */
XX X  01 H  XX X  XX X  XX X  XX X
XX X  10 H  XX X  XX X  XX X  XX X
XX X  XX X  00 H  XX X  XX X  XX X  /* test NAND gate */
XX X  XX X  01 H  XX X  XX X  XX X
XX X  XX X  11 L  XX X  XX X  XX X
XX X  XX X  XX X  00 H  XX X  XX X  /* test NOR gate */
XX X  XX X  XX X  01 L  XX X  XX X
XX X  XX X  XX X  XX X  00 L  XX X  /* test XOR gate */
XX X  XX X  XX X  XX X  01 H  XX X
XX X  XX X  XX X  XX X  10 H  XX X
XX X  XX X  XX X  XX X  XX X  00 H  /* test XNOR gate */
XX X  XX X  XX X  XX X  XX X  01 L
XX X  XX X  XX X  XX X  XX X  11 H

```

Figure 5. JEDEC File

```

Location      U4ia
*QP20
*QF2194
*G0
*F0
*L0256 11011101111111111111111111111111
*L0512 01111111111111111111111111111111
*L0544 11110111111111111111111111111111
*L0768 11111110111011111111111111111111
*L1024 11111111111111110111111111111111
*L1056 11111111111111111101111111111111
*L1280 1111111111111111111111101111011
*L1312 111111111111111111111110110111
*L1536 11111111111111111111111111101101
*L1568 111111111111111111111110111101
*L2048 01100100000000000000000000000000
*L2112 00000000100000001111111111111111
*L2144 11111111111111111111111111111111
*L2176 1111111111111111110
*C2F02
*V0001 0XXXXXXXXXXXXXXXXXN
*V0002 1XXXXXXXXXXXXXXXXXN
*V0003 1XXXXXXXXXXXXXXXXXN
*V0004 X00XXXXXXXXXXXXXXXXXN
*V0005 X01XXXXXXXXXXXXXXXXXN
*V0006 X10XXXXXXXXXXXXXXXXXN
*V0007 XXXX00XXXXXXXXXXXXXXXXXN
*V0008 XXXX01XXXXXXXXXXXXXXXXXN
*V0009 XXXX11XXXXXXXXXXXXXXXXXN
*V0010 XXXXXX00XXNXXXXXXXXXN
*V0011 XXXXXX01XXNXXXXXXXXXN
*V0012 XXXXXX10XXNXXXXXXXXXN
*V0013 XXXXXXXX00NXXNXXXXXN
*V0014 XXXXXXXX01NXXNXXXXXN
*V0015 XXXXXXXX10NXXNXXXXXN
*V0016 XXXXXXXXXXN00HXXXXXN
*V0017 XXXXXXXXXXN01LXXXXXN
*V0018 XXXXXXXXXXN11HXXXXXN
*4E79

```

Figure 6. CUPL Pinout Diagram

```

*****
*
* *
*
****
B * 1
****
*
****
C * 2
****
*
****
D * 3
****
*
****
E * 4
****
*
****
F * 5
****
*
****
G * 6
****
*
****
H * 7
****
*
****
I * 8
****
*
****
J * 9
****
*
****
GND * 10
****
*
*****

*****
*
*
*
****
20 * VCC
****
*
****
19 * A
****
*
****
18 * U
****
*
****
17 * V
****
*
****
16 * W
****
*
****
15 * X
****
*
****
14 * Y
****
*
****
13 * Z
****
*
****
12 * L
****
*
****
11 * K
****
*
*****

```

Figure 7. 'Fuse' Plot

Syn 2192 - Ac0 2193 x

Pin #19 2048 Pol x 2120 Ac1 -

```

0000 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0032 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0064 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0096 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0128 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0160 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0192 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0224 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

```

Pin #18 2049 Pol - 2121 Ac1 x

```

0256 --x--x-----
0288 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0320 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0352 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0384 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0416 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0448 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0480 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

```

Pin #17 2050 Pol - 2122 Ac1 x

```

0512 x-----
0544 ----x-----
0576 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0608 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0640 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0672 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0704 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0736 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

```

Pin #16 2051 Pol x 2123 Ac1 x

```

0768 -----x--x-----
0800 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0832 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0864 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0896 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0928 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0960 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0992 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

```

Pin #15 2052 Pol x 2124 Ac1 x

```

1024 -----x-----
1056 -----x-----
1088 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1120 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1152 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1184 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1216 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1248 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

```

Pin #14 2053 Pol - 2125 Ac1 x

```

1280 -----x--x-----
1312 -----x--x-----
1344 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1376 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1408 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1440 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1472 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1504 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

```

Pin #13 2054 Pol x 2126 Ac1 x

```

1536 -----x--x-----
1568 -----x--x-----
1600 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1632 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1664 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1696 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1728 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1760 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

```

Pin #12 2055 Pol x 2127 Ac1 -

```

1792 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1824 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1856 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1888 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1920 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1952 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1984 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
2016 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

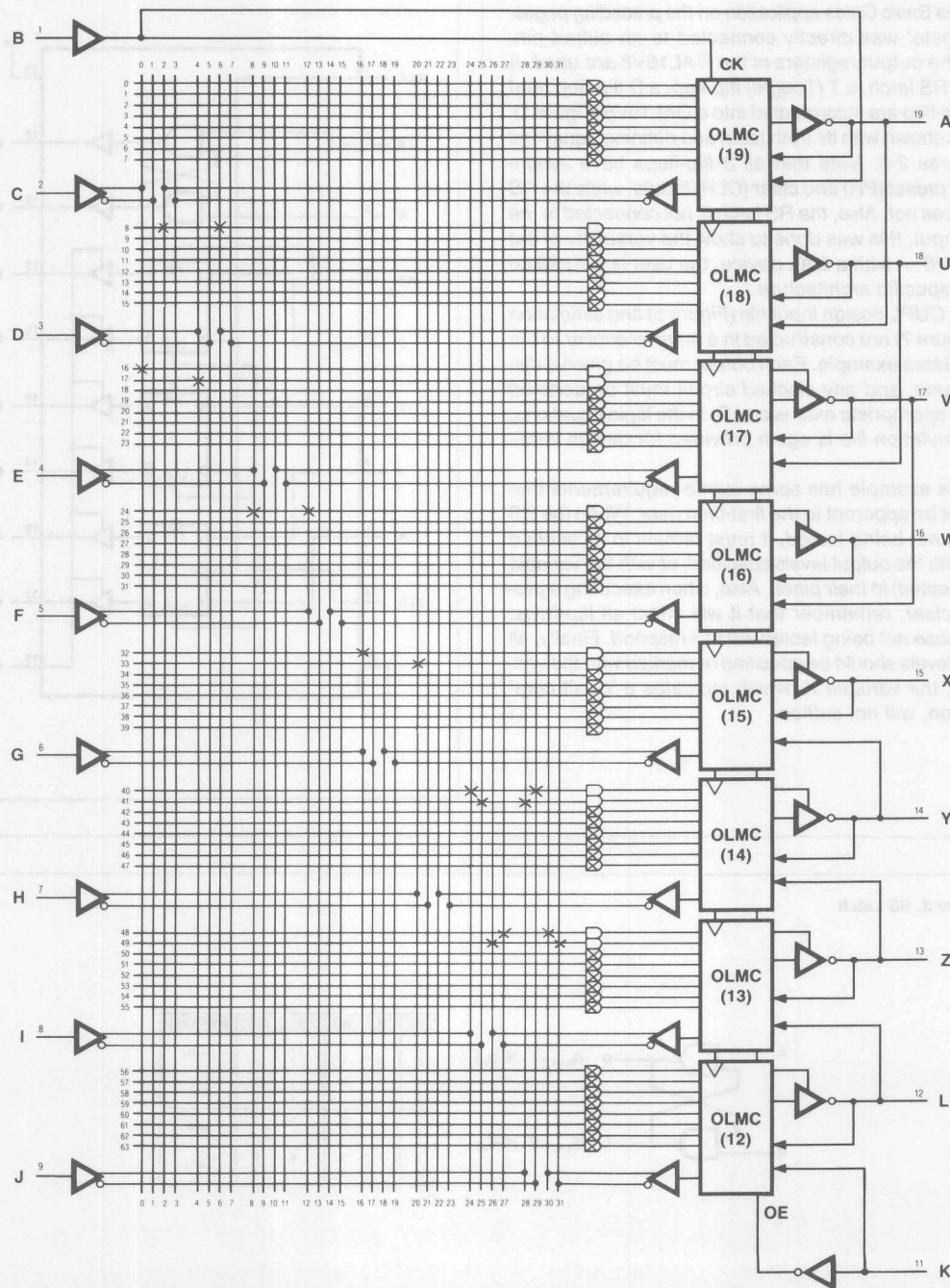
```

## LEGEND

X : Programmed Cell

- : Erased Cell

Figure 8. GAL16V8 Logic Diagram





## DESCRIPTION

In the Basic Gates application on the preceding pages, each 'gate' was directly connected to an output pin. Here, the output registers of the GAL16V8 are used. A simple RS latch, a T (Toggle) flip-flop, a D flip-flop, and a JK flip-flop are incorporated into a GAL16V8 (Figure 1). Each is shown with its truth table and defining equations in Figures 2-5. Note that all 3 flip-flops have synchronous preset (PR) and clear (CLR) inputs, while the RS latch does not. Also, the RS latch is not connected to the clock input; this was done to show the versatility of the GAL16V8 — with a GAL device, the user is not locked in to a specific architecture.

The CUPL design input file (Figure 6) and simulation file (Figure 7) are constructed in a manner similar to the Basic Gates example. Each output must be given a distinct name, and any clocked circuit must be denoted with an appropriate extension (.D) in the logic equations. The simulation file is again provided for design verification.

This example has some subtle requirements that may not be apparent to the first-time user. When the RS latch is not being tested, it must remain in its latched state with the output levels specified, or with the variable N (not tested) in their place. Also, when executing a preset or clear, remember that it will affect all flip-flops; even those not being tested will still respond. Finally, all output levels should be specified or marked with the variable N; the variable X, which indicates a 'don't care' condition, will not suffice. □

Figure 1. Basic Flip-Flops Pinout

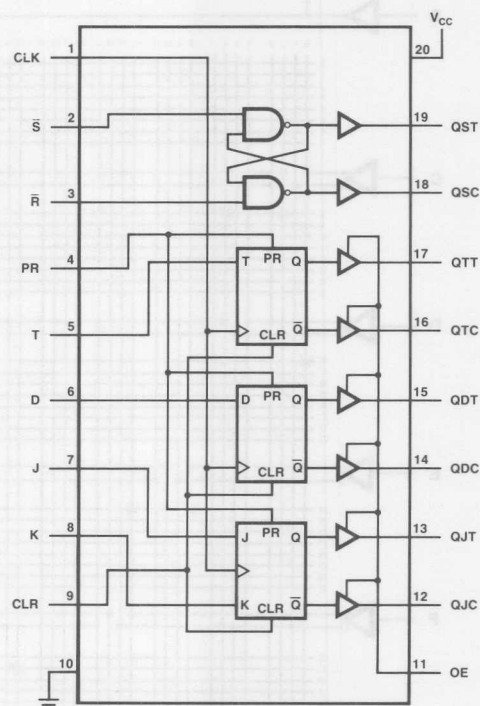
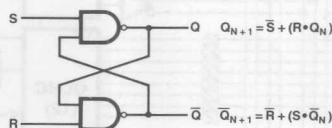


Figure 2. RS Latch



S	R	$Q_N$	$Q_{N+1}$	$\bar{Q}_{N+1}$	COMMENTS
0	0	0	1	1	Invalid
0	0	1	1	1	
0	1	0	1	0	Set
0	1	1	1	0	
1	0	0	0	1	Reset
1	0	1	0	1	
1	1	0	0	1	Latch
1	1	1	1	0	

Figure 3. T Flip-Flop

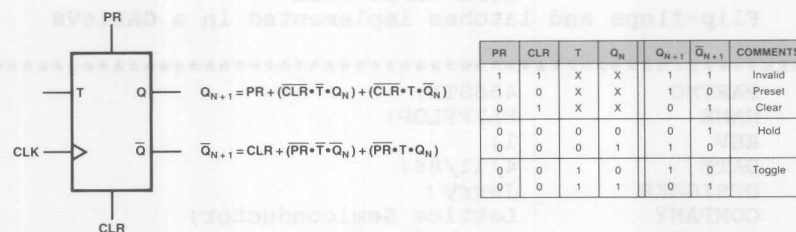


Figure 4. D Flip-Flop

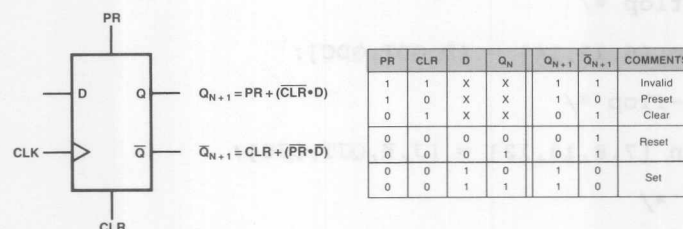


Figure 5. JK Flip-Flop

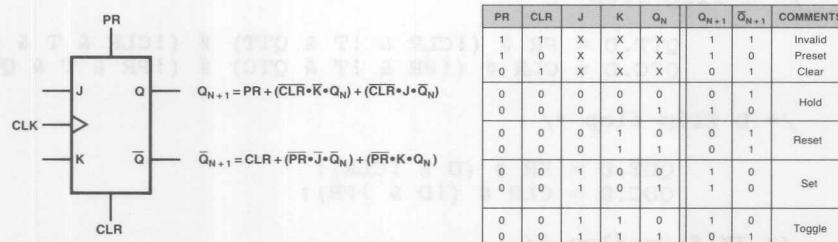


Figure 6. CUPL Input File

```

/*****
/*
/*          CUPL INPUT FILE          */
/*      Flip-flops and latches implemented in a GAL16V8      */
/*
/*
/*****
PARTNO      456STX;
NAME        FLIPFLOP;
REV         1;
DATE        4/11/86;
DESIGNER     Jerry;
COMPANY      Lattice Semiconductor;
ASSEMBLY     Clock Board;
LOCATION      U238;

/* RS latch */

    pin [2,3,19,18] = [S,R,QST,QSC];

/* T flip-flop */

    pin [5,17,16] = [T,QT,QT];

/* D flip-flop */

    pin [6,15,14] = [D,QDT,QDC];

/* JK flip-flop */

    pin [7,8,13,12] = [J,K,QJT,QJC];

/* control */

    pin [1,4,9,11] = [CLK,PR,CLR,OE];

/* logic equations */

/* RS latch */

    QST = !S # (R & QST);
    QSC = !R # (S & QSC);

/* T flip-flop */

    QT.D = PR # (!CLR & !T & QT) # (!CLR & T & QT);
    QT.D = CLR # (!PR & !T & QT) # (!PR & T & QT);

/* D flip-flop */

    QDT.D = PR # (D & !CLR);
    QDC.D = CLR # (!D & !PR);

/* JK flip-flop */

    QJT.D = PR # (J & QJC & !CLR) # (!K & QJT & !CLR);
    QJC.D = CLR # (!J & QJC & !PR) # (K & QJT & !PR);

```

Figure 7. CUPL Simulation File

```

/*****
/*
/*          CUPL SIMULATION FILE
/*      Flip-flops and latches implemented in a GAL16V8
/*
/*
*****/

PARTNO      456STX;
NAME        FLIPFLOP;
REV         1;
DATE        4/11/86;
DESIGNER    Jerry;
COMPANY     Lattice Semiconductor;
ASSEMBLY    Clock Board;
LOCATION     U238;

/* The Order statement specifies the layout of the vector table.
   %n = n spaces inserted between variables.
*/

order:  OE,%1,CLK,%2,S,R,%1,QST,QSC,%2,PR,%1,CLR,%2,
        T,%1,QTT,QTC,%2,D,%1,QDT,QDC,%2,J,K,%1,QJT,QJC;

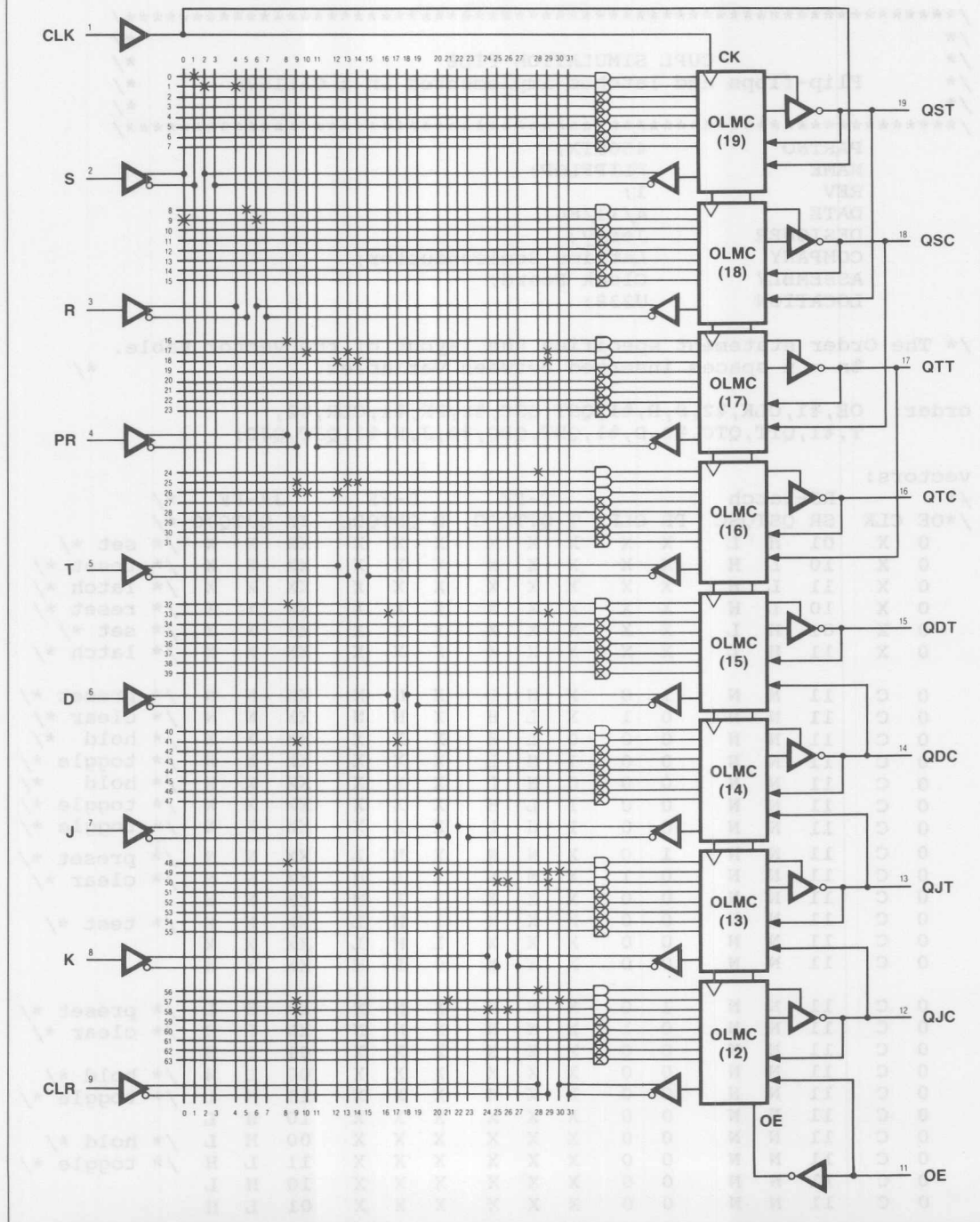
vectors:
/*      RS-latch      T-FF      D-FF      JK-FF      */
/*OE CLK  SR QSTQSC  PR CLR  T QTTQTC  D QDTQDC  JK QJTQJC */
0 X  01 H  L      X X  X X X  X X X  XX X X /* set */
0 X  10 L  H      X X  X X X  X X X  XX X X /* reset */
0 X  11 L  H      X X  X X X  X X X  XX X X /* latch */
0 X  10 L  H      X X  X X X  X X X  XX X X /* reset */
0 X  01 H  L      X X  X X X  X X X  XX X X /* set */
0 X  11 H  L      X X  X X X  X X X  XX X X /* latch */

0 C  11 N  N      1 0  X H L  X N N  XX N N /* preset */
0 C  11 N  N      0 1  X L H  X N N  XX N N /* clear */
0 C  11 N  N      0 0  0 L H  X X X  XX X X /* hold */
0 C  11 N  N      0 0  1 H L  X X X  XX X X /* toggle */
0 C  11 N  N      0 0  0 H L  X X X  XX X X /* hold */
0 C  11 N  N      0 0  1 L H  X X X  XX X X /* toggle */
0 C  11 N  N      0 0  1 H L  X X X  XX X X /* toggle */
0 C  11 N  N      1 0  X N N  X H L  XX N N /* preset */
0 C  11 N  N      0 1  X N N  X L H  XX N N /* clear */
0 C  11 N  N      0 0  X X X  0 L H  XX X X /* test */
0 C  11 N  N      0 0  X X X  1 H L  XX X X
0 C  11 N  N      0 0  X X X  0 L H  XX X X

0 C  11 N  N      1 0  X N N  X N N  XX H L /* preset */
0 C  11 N  N      0 1  X N N  X N N  XX L H /* clear */
0 C  11 N  N      0 0  X X X  X X X  01 L H
0 C  11 N  N      0 0  X X X  X X X  00 L H /* hold */
0 C  11 N  N      0 0  X X X  X X X  11 H L /* toggle */
0 C  11 N  N      0 0  X X X  X X X  10 H L
0 C  11 N  N      0 0  X X X  X X X  00 H L /* hold */
0 C  11 N  N      0 0  X X X  X X X  11 L H /* toggle */
0 C  11 N  N      0 0  X X X  X X X  10 H L
0 C  11 N  N      0 0  X X X  X X X  01 L H

```

Figure 8. GAL16V8 Logic Diagram





## DESCRIPTION

Shift registers are used in communications circuits and for arithmetic functions. This 6-bit shift register, built with the GAL16V8, can shift data left or right, and load data in serial or parallel manner (Figure 1). Two inputs control the mode,  $S_0$  on pin 2 and  $S_1$  on pin 3, in accordance with the function table in Figure 2. The serial ports are bidirectional, thus pin 12 is the serial-right input and serial-left output (RI/LO), while pin 19 is the serial-left input and serial-right output (LI/RO). The pinout diagram is shown in Figure 3. As shown in the circuit diagram of Figure 4, the shift register integrates a respectable amount of gates, yet the CUPL programming language makes its implementation straightforward. The design input file is shown in Figure 5, and simulation file in Figure 6.

Figure 1. Block Diagram

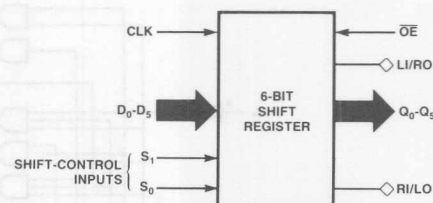


Figure 2. Mode Function Table

OE	S <sub>1</sub>	S <sub>0</sub>	LI/RO	Q <sub>5</sub>	Q <sub>4</sub>	Q <sub>3</sub>	Q <sub>2</sub>	Q <sub>1</sub>	Q <sub>0</sub>	LI/RO	COMMENTS
1	X	X	X	Z	Z	Z	Z	Z	Z	X	HIGH-Z OUTPUTS
0	0	0	X	Q <sub>5</sub>	Q <sub>4</sub>	Q <sub>3</sub>	Q <sub>2</sub>	Q <sub>1</sub>	Q <sub>0</sub>	X	HOLD
0	0	1	Q <sub>5</sub>	Q <sub>4</sub>	Q <sub>3</sub>	Q <sub>2</sub>	Q <sub>1</sub>	Q <sub>0</sub>	D <sub>0</sub>	D	SHIFT LEFT
0	1	0	D	D	Q <sub>5</sub>	Q <sub>4</sub>	Q <sub>3</sub>	Q <sub>2</sub>	Q <sub>1</sub>	Q <sub>0</sub>	SHIFT RIGHT
0	1	1	X	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>	X	LOAD

Figure 3. Shift Register Pinout

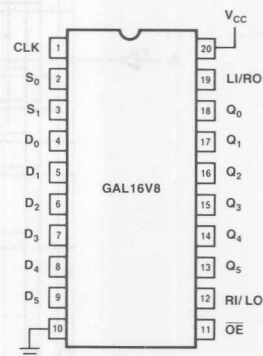


Figure 4. Shift Register Logic Diagram

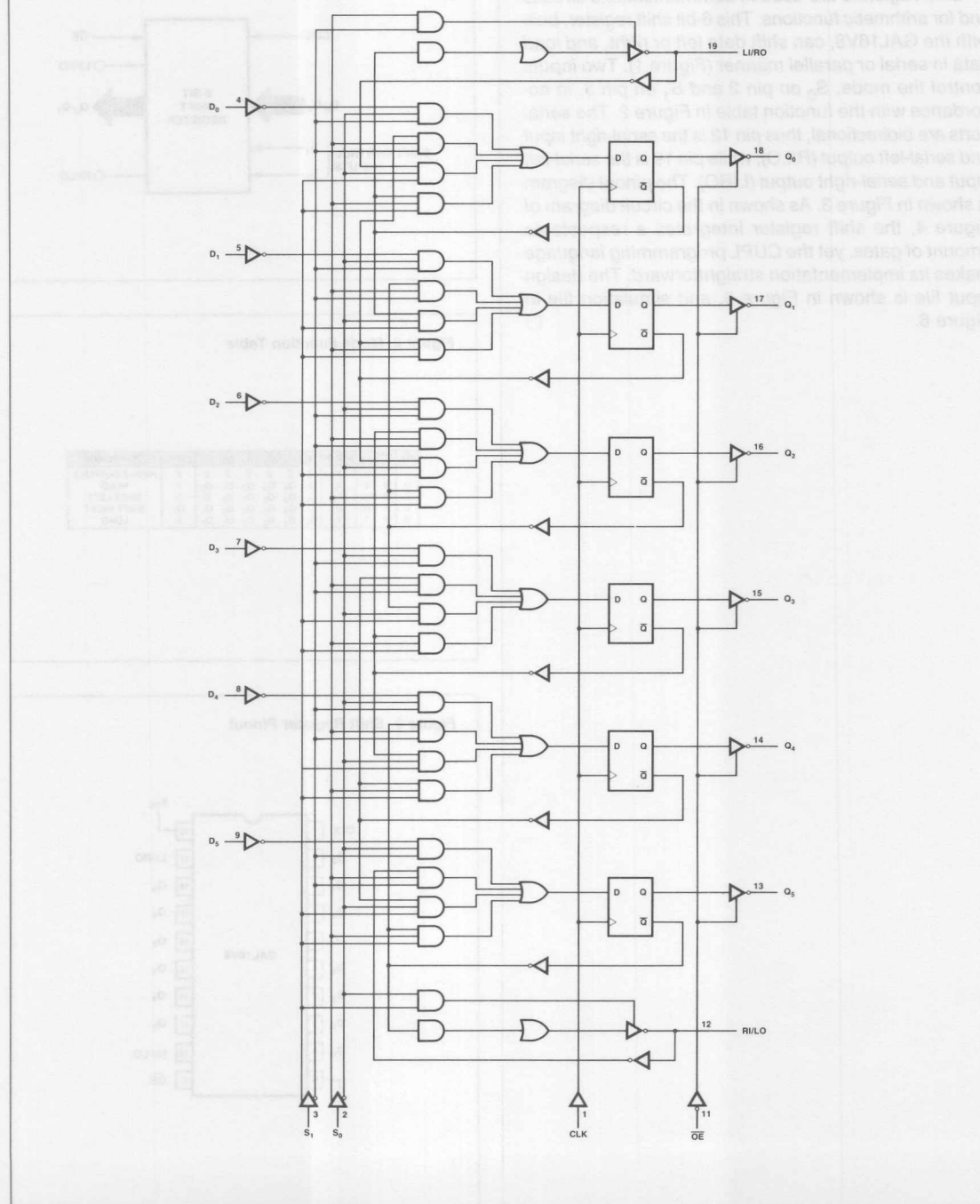


Figure 5. CUPL Design Input File

```

/*****
/*
/*          CUPL INPUT FILE          */
/*      6-bit shift register implemented in a GAL16V8      */
/*
/*
*****/

PARTNO      PIPO;
NAME        SHFTREG6;
REV         1;
DATE        4/15/86;
DESIGNER    Joe Eng;
COMPANY     Lattice Semiconductor;
ASSEMBLY    Communications;
LOCATION     U2;

/* inputs */

pin [4..9] = [D0..D5];
pin [2,3]  = [S0,S1];
pin [12,19] = [RILO,LIRO];
pin 1      = CLK;
pin 11     = OE;

/* outputs */

pin [18..13] = [Q0..Q5];

/* logic equations */

!Q0.D = (!S0 & !S1 & !Q0) # (S0 & !S1 & !Q1) #
        (!S0 & S1 & !LIRO) # (S0 & S1 & !D0);

!Q1.D = (!S0 & !S1 & !Q1) # (S0 & !S1 & !Q2) #
        (!S0 & S1 & !Q0) # (S0 & S1 & !D1);

!Q2.D = (!S0 & !S1 & !Q2) # (S0 & !S1 & !Q3) #
        (!S0 & S1 & !Q1) # (S0 & S1 & !D2);

!Q3.D = (!S0 & !S1 & !Q3) # (S0 & !S1 & !Q4) #
        (!S0 & S1 & !Q2) # (S0 & S1 & !D3);

!Q4.D = (!S0 & !S1 & !Q4) # (S0 & !S1 & !Q5) #
        (!S0 & S1 & !Q3) # (S0 & S1 & !D4);

!Q5.D = (!S0 & !S1 & !Q5) # (S0 & !S1 & !RILO) #
        (!S0 & S1 & !Q4) # (S0 & S1 & !D5);

```

Figure 6. CUPL Simulation File

```

/*****
/*
/*          CUPL SIMULATION FILE          */
/*          6-bit shift register implemented in a GAL16V8          */
/*
*****/

PARTNO          PIPO;
NAME            SHFTREG6;
REV            1;
DATE           4/15/86;
DESIGNER       Joe Eng;
COMPANY        Lattice Semiconductor;
ASSEMBLY       Communications;
LOCATION        U2;

order:  S0,%1,S1,%2,D5,%1,D4,%1,D3,%1,D2,%1,D1,%1,D0,%2,RILO,%1,
        LIRO,%2,Q5,%1,Q4,%1,Q3,%1,Q2,%1,Q1,%1,Q0,%2,CLK,%1,OE;

vectors:

/*  S      D      RI LI      Q      */
/*  0 1    5 4 3 2 1 0  LO RO 5 4 3 2 1 0  C OE */
1 1    0 0 0 0 0 0    X X  L L L L L L  C 0  /* load 0s */
0 0    X X X X X X    X X  L L L L L L  C 0  /* hold */
1 1    1 1 1 1 1 1    X X  H H H H H H  C 0  /* load 1s */
0 0    X X X X X X    X X  H H H H H H  C 0  /* hold */
0 1    X X X X X X    1 0  H H H H H L  C 0  /* shift left */
0 1    X X X X X X    1 1  H H H H L H  C 0
0 1    X X X X X X    1 1  H H H L H H  C 0
0 1    X X X X X X    1 1  H H L H H H  C 0
0 1    X X X X X X    1 1  H L H H H H  C 0
0 1    X X X X X X    0 1  L H H H H H  C 0
0 1    X X X X X X    1 1  H H H H H H  C 0
1 0    X X X X X X    0 1  L H H H H H  C 0  /* shift right */
1 0    X X X X X X    1 1  H L H H H H  C 0
1 0    X X X X X X    1 1  H H L H H H  C 0
1 0    X X X X X X    1 1  H H H L H H  C 0
1 0    X X X X X X    1 0  H H H H H L  C 0
1 0    X X X X X X    1 1  H H H H H H  C 0

```

### DESCRIPTION

Here, a 4-bit synchronous up/down counter (Figure 1) implemented in a GAL16V8 provides additional functions of clear and load, which allow the counter to be reset to zero or loaded with any arbitrary value at any time. The pinout diagram is shown in Figure 2.

This application was developed with ABEL programming software, and is self-documenting. The truth table for the two control inputs (CNTL<sub>0</sub> and CNTL<sub>1</sub>) that determine the mode of the counter, for example, can be found in the design input file (Figure 3). Unlike CUPL, ABEL programming software provides input and output statements in one file; in this example, the listings have been separated for clarity. Input equations are listed in Figure 4, and test vectors for the counter, also provided by ABEL software, are shown in Figure 5. □

Figure 1. Block Diagram

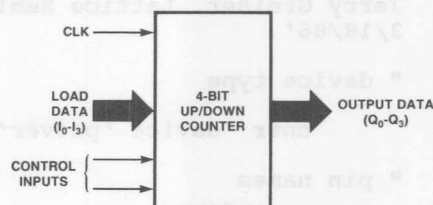


Figure 2. Pinout Diagram

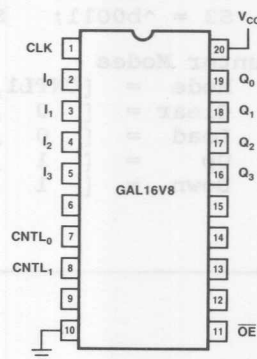




Figure 3. ABEL Input File

```

module counter
title 'ABEL INPUT FILE
4-bit Up/Down counter with Clear
Jerry Greiner, Lattice Semiconductor
3/18/86'

" device type
    cntr device 'pl6v8r';

" pin names
    CLK, CNTL0, CNTL1, OE    pin 1,7,8,11;
    IO,I1,I2,I3              pin 2,3,4,5;
    Q3,Q2,Q1,Q0              pin 16,17,18,19;

" constants
    C,X,Z      =      .C.,.X.,.Z.;

" Counter States
    S0 = ^b0000;   S4 = ^b0100;   S8 = ^b1000;   S12 = ^b1100;
    S1 = ^b0001;   S5 = ^b0101;   S9 = ^b1001;   S13 = ^b1101;
    S2 = ^b0010;   S6 = ^b0110;   S10 = ^b1010;   S14 = ^b1110;
    S3 = ^b0011;   S7 = ^b0111;   S11 = ^b1011;   S15 = ^b1111;

" Counter Modes
    Mode = [CNTL1,CNTL0 ];
    Clear = [ 0 , 0 ];
    Load = [ 0 , 1 ];
    Up = [ 1 , 1 ];
    Down = [ 1 , 0 ];

```

Figure 4. ABEL Design Input File

equations

```
[Q3,Q2,Q1,Q0] := (Mode == Load) & [I3,I2,I1,I0];
```

```
Q3 := (CNTL0 & !CNTL1 & I3
# !CNTL0 & CNTL1 & !Q0 & !Q1 & !Q2 & !Q3
# !CNTL0 & CNTL1 & Q0 & Q1 & Q2 & Q3
# CNTL0 & CNTL1 & !Q0 & !Q1 & !Q2 & Q3
# CNTL0 & CNTL1 & Q0 & Q1 & Q2 & !Q3
# CNTL1 & Q0 & !Q2 & Q3
# CNTL1 & !Q1 & Q2 & Q3
# CNTL1 & !Q0 & Q1 & Q3);
```

```
Q2 := (CNTL0 & !CNTL1 & I2
# !CNTL0 & CNTL1 & !Q0 & !Q1 & !Q2
# !CNTL0 & CNTL1 & Q1 & Q2
# CNTL0 & CNTL1 & !Q1 & Q2
# CNTL1 & !Q0 & Q1 & Q2
# CNTL1 & Q0 & !Q1 & Q2
# CNTL0 & CNTL1 & Q0 & Q1 & !Q2);
```

```
Q1 := (CNTL0 & !CNTL1 & I1
# !CNTL0 & CNTL1 & !Q0 & !Q1
# !CNTL0 & CNTL1 & Q0 & Q1
# CNTL0 & CNTL1 & !Q0 & Q1
# CNTL0 & CNTL1 & Q0 & !Q1);
```

```
Q0 := (CNTL0 & !CNTL1 & I0 # CNTL1 & !Q0);
```

Figure 5. ABEL File Test Vectors

```

test_vectors ( [CLK, OE, I3, I2, I1, I0, Mode] -> [Q3, Q2, Q1, Q0] )
[C , 0,  X, X, X, X, Clear] -> S0;
[C , 0,  X, X, X, X, Up   ] -> S1;
[C , 0,  X, X, X, X, Up   ] -> S2;
[C , 0,  X, X, X, X, Up   ] -> S3;
[C , 0,  X, X, X, X, Up   ] -> S4;
[C , 0,  X, X, X, X, Up   ] -> S5;
[C , 0,  X, X, X, X, Up   ] -> S6;
[C , 0,  X, X, X, X, Up   ] -> S7;
[C , 0,  X, X, X, X, Up   ] -> S8;
[C , 0,  X, X, X, X, Up   ] -> S9;
[C , 0,  X, X, X, X, Up   ] -> S10;
[C , 0,  X, X, X, X, Up   ] -> S11;
[C , 0,  X, X, X, X, Up   ] -> S12;
[C , 0,  X, X, X, X, Up   ] -> S13;
[C , 0,  X, X, X, X, Up   ] -> S14;
[C , 0,  X, X, X, X, Up   ] -> S15;
[C , 0,  X, X, X, X, Up   ] -> S0;
[C , 0,  X, X, X, X, Up   ] -> S1;
[C , 0,  X, X, X, X, Down ] -> S0;
[C , 0,  X, X, X, X, Down ] -> S15;
[C , 0,  X, X, X, X, Down ] -> S14;
[C , 0,  X, X, X, X, Down ] -> S13;
[C , 0,  X, X, X, X, Down ] -> S12;
[C , 0,  X, X, X, X, Down ] -> S11;
[C , 0,  X, X, X, X, Down ] -> S10;
[C , 0,  X, X, X, X, Down ] -> S9;
[C , 0,  X, X, X, X, Down ] -> S8;
[C , 0,  X, X, X, X, Down ] -> S7;
[C , 0,  X, X, X, X, Down ] -> S6;
[C , 0,  X, X, X, X, Down ] -> S5;
[C , 0,  X, X, X, X, Down ] -> S4;
[C , 0,  X, X, X, X, Down ] -> S3;
[C , 0,  X, X, X, X, Down ] -> S2;
[C , 0,  X, X, X, X, Down ] -> S1;
[C , 0,  X, X, X, X, Down ] -> S0;
[C , 0,  X, X, X, X, Down ] -> S15;
[C , 0,  X, X, X, X, Down ] -> S14;
[C , 0,  0, 0, 0, 0, Load ] -> S0;
[C , 0,  1, 1, 1, 1, Load ] -> S15;
[C , 0,  1, 0, 1, 0, Load ] -> S10;

end counter

```

### DESCRIPTION

In this example, a GAL20V8 implements a seven-bit counter with asynchronous carry-out and load functions. As illustrated in the block diagram (Figure 1) and pinout diagram (Figure 2), the carry-in and carry-out pins make the counter fully cascadable to form larger counters. The CUPL design input files are shown in Figure 3, and simulation files in Figure 4. Note that the counter requires seven registers and one asynchronous output, taking full advantage of the generic architecture of the GAL20V8.

□

Figure 1. Block Diagram

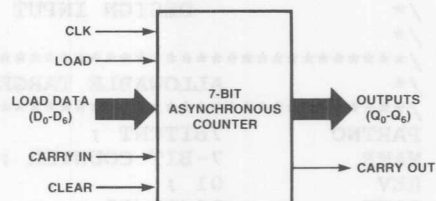


Figure 2. Pinout Diagram

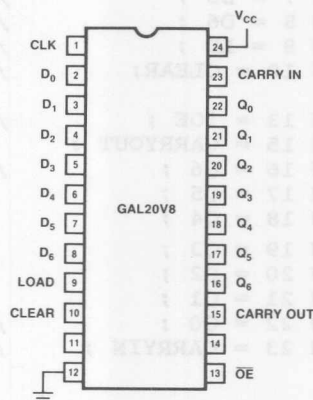


Figure 3. CUPL Design Input File

```

/*****
/*
/*          CUPL INPUT FILE
/*          DESIGN INPUT FOR 7-BIT COUNTER
/*
/*****
/*          ALLOWABLE TARGET DEVICE:  GAL20V8
/*****
PARTNO      7BITCNT ;
NAME        7-BIT COUNTER ;
REV         01 ;
DATE        10/08/85 ;
DESIGNER     JERRY GREINER ;
COMPANY      LATTICE SEMICONDUCTOR ;
ASSEMBLY     3A-27 ;
LOCATION      U06 ;

PIN 1 = CLK ;          /* CLOCK INPUT */
PIN 2 = D0 ;           /* DATA0 INPUT */
PIN 3 = D1 ;           /* DATA1 INPUT */
PIN 4 = D2 ;           /* DATA2 INPUT */
PIN 5 = D3 ;           /* DATA3 INPUT */
PIN 6 = D4 ;           /* DATA4 INPUT */
PIN 7 = D5 ;           /* DATA5 INPUT */
PIN 8 = D6 ;           /* DATA6 INPUT */
PIN 9 = LD ;           /* LOAD CONTROL */
PIN 10 = CLEAR;        /* ASYNCHRONOUS CARRY-IN */

PIN 13 = !OE ;         /* OUTPUT ENABLE */
PIN 15 = CARRYOUT ;
PIN 16 = Q6 ;          /* COUNTER MSB */
PIN 17 = Q5 ;
PIN 18 = Q4 ;
PIN 19 = Q3 ;
PIN 20 = Q2 ;
PIN 21 = Q1 ;
PIN 22 = Q0 ;          /* COUNTER LSB */
PIN 23 = CARRYIN ;     /* CARRY-IN FOR CASCADING */

```



Figure 3. (cont'd)

```

Q0.D = (LD & D0                                     /* LOAD D0 */
# !LD & !Q0 & CARRYIN) & !CLEAR;                    /* TOGGLE */

Q1.D = (LD & D1                                     /* LOAD D1 */
# !LD& !Q1 & Q0 & CARRYIN                             /* TOGGLE */
# !LD& Q1 & !Q0) & !CLEAR;                          /* HOLD */

Q2.D = (LD & D2                                     /* LOAD D2 */
# !LD& !Q2 & Q1 & Q0 & CARRYIN                       /* TOGGLE */
# !LD& Q2 & !Q1                                       /* HOLD */
# !LD& Q2 & !Q0) & !CLEAR;                          /* HOLD */

Q3.D = (LD & D3                                     /* LOAD D3 */
# !LD& !Q3 & Q2 & Q1 & Q0 & CARRYIN                   /* TOGGLE */
# !LD& Q3 & !Q2                                       /* HOLD */
# !LD& Q3 & !Q1                                       /* HOLD */
# !LD& Q3 & !Q0) & !CLEAR;                          /* HOLD */

Q4.D = (LD & D4                                     /* LOAD D4 */
# !LD& !Q4& Q3 & Q2 & Q1 & Q0 & CARRYIN               /* TOGGLE */
# !LD& Q4 & !Q3                                       /* HOLD */
# !LD& Q4 & !Q2                                       /* HOLD */
# !LD& Q4 & !Q1                                       /* HOLD */
# !LD& Q4 & !Q0) & !CLEAR;                          /* HOLD */

Q5.D = (LD & D5                                     /* LOAD D5 */
# !LD& !Q5& Q4 & Q3 & Q2 & Q1 & Q0
& CARRYIN                                           /* TOGGLE */
# !LD& Q5 & !Q4                                       /* HOLD */
# !LD& Q5 & !Q3                                       /* HOLD */
# !LD& Q5 & !Q2                                       /* HOLD */
# !LD& Q5 & !Q1                                       /* HOLD */
# !LD& Q5 & !Q0) & !CLEAR;                          /* HOLD */

Q6.D = (LD & D6                                     /* LOAD D6 */
# !LD& !Q6& Q5 & Q4 & Q3 & Q2 & Q1 & Q0
& CARRYIN                                           /* TOGGLE */
# !LD& Q6 & !Q5                                       /* HOLD */
# !LD& Q6 & !Q4                                       /* HOLD */
# !LD& Q6 & !Q3                                       /* HOLD */
# !LD& Q6 & !Q2                                       /* HOLD */
# !LD& Q6 & !Q1                                       /* HOLD */
# !LD& Q6 & !Q0) & !CLEAR;                          /* HOLD */

CARRYOUT = !LD & Q6 & Q5 & Q4 & Q3 & Q2 & Q1 & Q0
& CARRYIN;                                          /* CARRY-OUT */

```

Figure 4. CUPL Simulation File

```

/*****
/*
/*          CUPL INPUT FILE
/*          SIMULATION FOR 7-BIT COUNTER
/*
/*
/*          ALLOWABLE TARGET DEVICE: GAL20V8
/*
*****/
PARTNO      7BITCNT ;
NAME        7-BIT COUNTER ;
REV         01 ;
DATE        10/08/85 ;
DESIGNER    JERRY GREINER ;
COMPANY     LATTICE SEMICONDUCTOR ;
ASSEMBLY    3A-27 ;
LOCATION     U06 ;

ORDER:

CLK, !OE, CLEAR, LD, CARRYIN, D6, D5, D4, D3, D2, D1, D0, Q6,
Q5, Q4, Q3, Q2, Q1, Q0, CARRYOUT;

```

Figure 4. (cont'd)

## VECTORS:

```

$msg" C ! C C C " ;
$msg" L O L L I DDDDDDD QQQQQQQ U " ;
$msg" K E R D N 6543210 6543210 T " ;
$msg" ----- " ;
0 1 X X X XXXXXXX ZZZZZZZ X /* TEST HI-Z */
C 0 1 X X XXXXXXX LLLLLLL L /* TEST CLEAR */
C 0 0 1 X 1111111 HHHHHHH L /* LOAD ONES */
C 0 0 1 X 0000000 LLLLLLL L /* LOAD ZEROS */
C 0 0 0 1 XXXXXXX LLLLLLL L /* COUNT=1 */
C 0 0 0 1 XXXXXXX LLLLLHL L /* COUNT=2 */
C 0 0 0 1 XXXXXXX LLLLLHH L /* COUNT=3 */
C 0 0 0 1 XXXXXXX LLLHLHL L /* COUNT=4 */
C 0 0 0 1 XXXXXXX LLLHLHL L /* COUNT=5 */
C 0 0 0 1 XXXXXXX LLLHLHL L /* COUNT=6 */
C 0 0 0 1 XXXXXXX LLLHHHL L /* COUNT=7 */
C 0 0 0 1 XXXXXXX LLLHLLL L /* COUNT=8 */
C 0 0 0 1 XXXXXXX LLLHLLH L /* COUNT=9 */
C 0 0 0 1 XXXXXXX LLLHLHL L /* COUNT=10 */
C 0 0 0 1 XXXXXXX LLLHLHH L /* COUNT=11 */
C 0 0 0 1 XXXXXXX LLLHLLH L /* COUNT=12 */
C 0 0 0 1 XXXXXXX LLLHHLH L /* COUNT=13 */
C 0 0 0 1 XXXXXXX LLLHHHL L /* COUNT=14 */
C 0 0 0 1 XXXXXXX LLLHHHL L /* COUNT=15 */
C 0 0 0 1 XXXXXXX LLHLLLL L /* COUNT=16 */
C 0 0 0 1 XXXXXXX LLHLLHL L /* COUNT=17 */
C 0 0 0 1 XXXXXXX LLHLLHL L /* COUNT=18 */
C 0 0 0 1 XXXXXXX LLHLLHH L /* COUNT=19 */
C 0 0 0 1 XXXXXXX LLHLHLL L /* COUNT=20 */
C 0 0 0 1 XXXXXXX LLHLHLH L /* COUNT=21 */
C 0 0 0 1 XXXXXXX LLHLHHL L /* COUNT=22 */
C 0 0 0 1 XXXXXXX LLHLHHH L /* COUNT=23 */
C 0 0 0 1 XXXXXXX LLHLLLL L /* COUNT=24 */
C 0 0 0 1 XXXXXXX LLHLLHL L /* COUNT=25 */
C 0 0 0 1 XXXXXXX LLHHLHL L /* COUNT=26 */
C 0 0 0 1 XXXXXXX LLHHLHH L /* COUNT=27 */
C 0 0 0 1 XXXXXXX LLHHLLL L /* COUNT=28 */
C 0 0 0 1 XXXXXXX LLHHHLH L /* COUNT=29 */
C 0 0 0 1 XXXXXXX LLHHHHL L /* COUNT=30 */
C 0 0 0 1 XXXXXXX LLHHHHH L /* COUNT=31 */
C 0 0 0 1 XXXXXXX LHLLLLL L /* COUNT=32 */
C 0 0 0 1 XXXXXXX LHLLLLH L /* COUNT=33 */
C 0 0 1 X 0111111 LHHHHHH L /* LOAD=63 TO OBSERVE MSB TOGGLE */
C 0 0 0 1 XXXXXXX HLLLLLL L /* COUNT=64, OBSERVE MSB */
C 0 0 0 1 XXXXXXX HLLLLHL L /* COUNT=65, OBSERVE MSB */
C 0 0 1 X 1111110 HHHHHHL L /* LOAD=126 TO OBSERVE CARRY */
C 0 0 0 1 XXXXXXX HHHHHHH H /* COUNT=127, OBSERVE CARRY */
C 0 0 0 1 XXXXXXX LLLLLLL L /* COUNT=0, OBSERVE CARRY */

```

## DESCRIPTION

Address decoding is a very common application of programmable logic devices. This example has been reprinted (with a few modifications) from the "ABEL Applications Guide," with the permission of Data I/O Corporation, Redmond, WA. The purpose of this GAL16V8 decoder is to monitor the high-order bits of a 16-bit address bus and select the correct memory or I/O device based on the value of these address bits, as illustrated in Figure 1. The memory map is shown in Figure 2.

The logic design input file for the decoder is shown in Figure 3. Note that the program utilizes an intermediate variable, "Address," and relational operators to make the program more readable. Reduced equations provided by the software are shown in Figure 4, and ABEL's document generator draws the pinout diagram (Figure 5), as well. □

Figure 1. Block Diagram

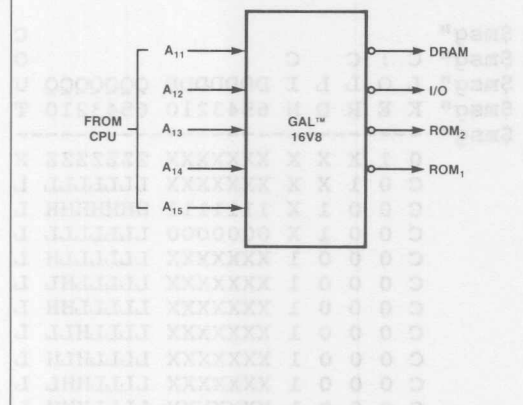


Figure 2. Memory Map

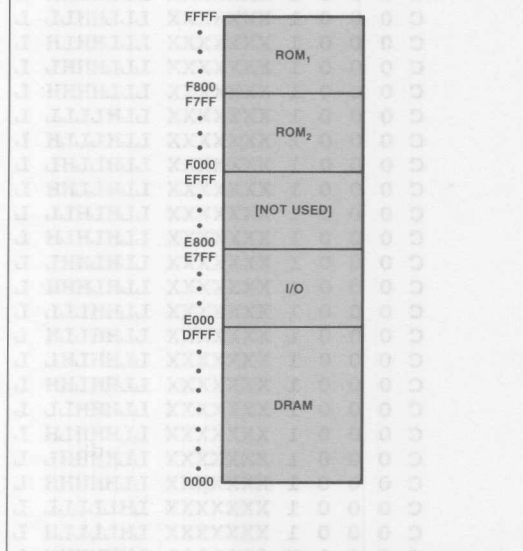


Figure 3. ABEL Input File

```

module  addrdec2

title  'ABEL INPUT FILE
      Address Decoder for a 6809 CPU System
      reprinted with permission from Data I/O Corp.
      Jerry                      April 8, 1986
      Lattice Semiconductor Corp.'
```

"device declaration

location	keyword	device code
U04	device	'P16V8S';

"pin declaration

A11,A12,A13,A14,A15	pin 2,3,4,5,6;
ROM1,ROM2,IO,DRAM	pin 16,17,18,19;

"constant declaration

```

H,L,X  = 1,0,.X.;
Address = [A15,A14,A13,A12, A11,X,X,X, X,X,X,X, X,X,X,X];
```

equations

```

!DRAM = (Address <= ^hDFFF);    "DRAM selected when Address
                                " less than or equal to DFFF

!IO   = (Address >= ^hE000) &    "IO selected when Address greater
      (Address <= ^hE7FF);      " than or equal to E000 but less
                                " than or equal to E7FF

!ROM2 = (Address >= ^hF000) &    "ROM2 selected when Address
      (Address <= ^hF7FF);      " greater than or equal to F000
                                " but less than or equal to F7FF

!ROM1 = (Address >= ^hF800);     "ROM1 selected when Address
                                " greater than or equal to F800
```

test\_vectors (Address -> [ROM1,ROM2,IO,DRAM])

^h0000	-> [ H, H, H, L ];	" select DRAM
^h5555	-> [ H, H, H, L ];	" "
^hAAAA	-> [ H, H, H, L ];	" "
^hDFFF	-> [ H, H, H, L ];	" "
^hE000	-> [ H, H, L, H ];	" select
^hE7FF	-> [ H, H, L, H ];	" IO
^hE800	-> [ H, H, H, H ];	" (not
^hEFFF	-> [ H, H, H, H ];	" used)
^hF000	-> [ H, L, H, H ];	" select
^hF7FF	-> [ H, L, H, H ];	" ROM2
^hF800	-> [ L, H, H, H ];	" select
^hFFFF	-> [ L, H, H, H ];	" ROM1

```

end addrdec2
```



Figure 4. Reduced Equations

ABEL(tm) Version 1.19 - Document Generator  
 Address Decoder for a 6809 CPU System  
 reprinted with permission from Data I/O Corp.  
 Jerry April 8, 1986  
 Lattice Semiconductor Corp.  
 Equations for Module addrdec2  
 Device U04

## Reduced Equations:

DRAM = (A13 & A14 & A15);

IO = !(A11 & A12 & A13 & A14 & A15);

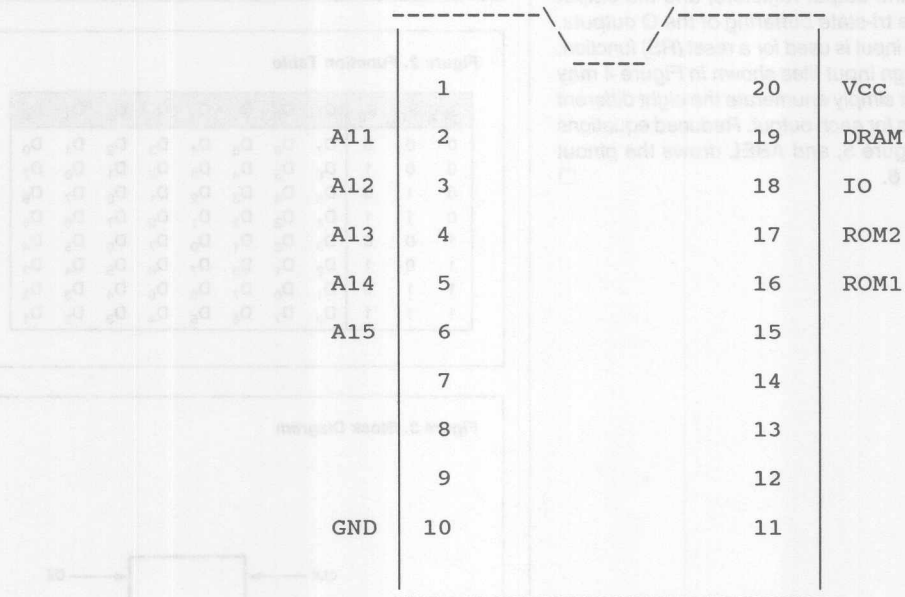
ROM2 = !(A11 & A12 & A13 & A14 & A15);

ROM1 = (!A15  
 # !A14 & A15  
 # !A13 & A14 & A15  
 # !A12 & A13 & A14 & A15  
 # !A11 & A12 & A13 & A14 & A15);

Figure 5. Pinout Diagram

ABEL(tm) Version 1.19 - Document Generator  
 Address Decoder for a 6809 CPU System  
 reprinted with permission from Data I/O Corp.  
 Jerry  
 Lattice Semiconductor Corp.  
 Chip diagram for Module addrdec2  
 Device U04

P16V8



end of module addrdec2

## DESCRIPTION

The barrel shifter (Figure 1) is a specialized shift register that rotates data a selectable number of bit positions out of the most-significant bit and back into the least-significant bit — thus the name. Typical applications of a barrel shifter are floating-point arithmetic and display rotation on a graphics terminal.

Since our barrel shifter has 8 data inputs and 8 registered outputs, as well as control signals, the GAL20V8 is the PLD of choice. The shift-select inputs ( $S_0, S_1, S_2$ ) determine the number of positions shifted, as described in the function table of Figure 2. The block diagram is shown in Figure 3. The clock (CLK) input gates input data synchronously to the output registers, and the output enable (OE) allows tri-state buffering of the Q outputs. The one remaining input is used for a reset (RS) function.

The ABEL design input files shown in Figure 4 may appear tedious, but simply enumerate the eight different bit-shift possibilities for each output. Reduced equations are provided in Figure 5, and ABEL draws the pinout diagram in Figure 6. □

Figure 1. Barrel Shift Rotation

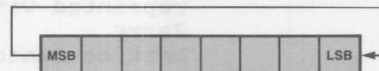


Figure 2. Function Table

$S_2$	$S_1$	$S_0$	$Q_7$	$Q_6$	$Q_5$	$Q_4$	$Q_3$	$Q_2$	$Q_1$	$Q_0$
0	0	0	$D_7$	$D_6$	$D_5$	$D_4$	$D_3$	$D_2$	$D_1$	$D_0$
0	0	1	$D_6$	$D_5$	$D_4$	$D_3$	$D_2$	$D_1$	$D_0$	$D_7$
0	1	0	$D_5$	$D_4$	$D_3$	$D_2$	$D_1$	$D_0$	$D_7$	$D_6$
0	1	1	$D_4$	$D_3$	$D_2$	$D_1$	$D_0$	$D_7$	$D_6$	$D_5$
1	0	0	$D_3$	$D_2$	$D_1$	$D_0$	$D_7$	$D_6$	$D_5$	$D_4$
1	0	1	$D_2$	$D_1$	$D_0$	$D_7$	$D_6$	$D_5$	$D_4$	$D_3$
1	1	0	$D_1$	$D_0$	$D_7$	$D_6$	$D_5$	$D_4$	$D_3$	$D_2$
1	1	1	$D_0$	$D_7$	$D_6$	$D_5$	$D_4$	$D_3$	$D_2$	$D_1$

Figure 3. Block Diagram

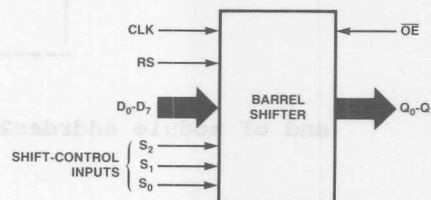


Figure 4. ABEL Input File

```

module barrel_shifter_8;

title 'ABEL INPUT FILE
      8-bit Barrel Shifter in a GAL20V8 April 16, 1986
      Lattice Semiconductor Joe Eng'

"device declaration
      "location keyword device code
      U9 device 'P20V8R' ;

"pin declaration

"inputs
D7,D6,D5,D4,D3,D2,D1,D0 pin 4,5,6,7,8,9,10,11;
CLK pin 1;

"outputs
Q7,Q6,Q5,Q4,Q3,Q2,Q1,Q0 pin 22,21,20,19,18,17,16,15;

"control
S2,S1,S0 pin 23,2,3; " selects 0-7 bit shift
RS pin 14; " resets all outputs to 0
OE pin 13; " output enable

"constant declaration

X = .X.; " simplify 'don't care' constant
C = .C.; " simplify 'clock' constant

```

Figure 4. (cont'd)

equations

```

Q0 := !RS & ((!S2 & !S1 & !S0 & D0) #
              (!S2 & !S1 & S0 & D7) #
              (!S2 & S1 & !S0 & D6) #
              (!S2 & S1 & S0 & D5) #
              (S2 & !S1 & !S0 & D4) #
              (S2 & !S1 & S0 & D3) #
              (S2 & S1 & !S0 & D2) #
              (S2 & S1 & S0 & D1));

Q1 := !RS & ((!S2 & !S1 & !S0 & D1) #
              (!S2 & !S1 & S0 & D0) #
              (!S2 & S1 & !S0 & D7) #
              (!S2 & S1 & S0 & D6) #
              (S2 & !S1 & !S0 & D5) #
              (S2 & !S1 & S0 & D4) #
              (S2 & S1 & !S0 & D3) #
              (S2 & S1 & S0 & D2));

Q2 := !RS & ((!S2 & !S1 & !S0 & D2) #
              (!S2 & !S1 & S0 & D1) #
              (!S2 & S1 & !S0 & D0) #
              (!S2 & S1 & S0 & D7) #
              (S2 & !S1 & !S0 & D6) #
              (S2 & !S1 & S0 & D5) #
              (S2 & S1 & !S0 & D4) #
              (S2 & S1 & S0 & D3));

Q3 := !RS & ((!S2 & !S1 & !S0 & D3) #
              (!S2 & !S1 & S0 & D2) #
              (!S2 & S1 & !S0 & D1) #
              (!S2 & S1 & S0 & D0) #
              (S2 & !S1 & !S0 & D7) #
              (S2 & !S1 & S0 & D6) #
              (S2 & S1 & !S0 & D5) #
              (S2 & S1 & S0 & D4));

Q4 := !RS & ((!S2 & !S1 & !S0 & D4) #
              (!S2 & !S1 & S0 & D3) #
              (!S2 & S1 & !S0 & D2) #
              (!S2 & S1 & S0 & D1) #
              (S2 & !S1 & !S0 & D0) #
              (S2 & !S1 & S0 & D7) #
              (S2 & S1 & !S0 & D6) #
              (S2 & S1 & S0 & D5));
    
```

Figure 4. (cont'd)

```

Q5 := !RS & ((!S2 & !S1 & !S0 & D5) #
          (!S2 & !S1 & S0 & D4) #
          (!S2 & S1 & !S0 & D3) #
          (!S2 & S1 & S0 & D2) #
          (S2 & !S1 & !S0 & D1) #
          (S2 & !S1 & S0 & D0) #
          (S2 & S1 & !S0 & D7) #
          (S2 & S1 & S0 & D6));

Q6 := !RS & ((!S2 & !S1 & !S0 & D6) #
          (!S2 & !S1 & S0 & D5) #
          (!S2 & S1 & !S0 & D4) #
          (!S2 & S1 & S0 & D3) #
          (S2 & !S1 & !S0 & D2) #
          (S2 & !S1 & S0 & D1) #
          (S2 & S1 & !S0 & D0) #
          (S2 & S1 & S0 & D7));

Q7 := !RS & ((!S2 & !S1 & !S0 & D7) #
          (!S2 & !S1 & S0 & D6) #
          (!S2 & S1 & !S0 & D5) #
          (!S2 & S1 & S0 & D4) #
          (S2 & !S1 & !S0 & D3) #
          (S2 & !S1 & S0 & D2) #
          (S2 & S1 & !S0 & D1) #
          (S2 & S1 & S0 & D0));

test_vectors ([CLK,OE,RS,S2,S1,S0,D7..D0] -> [Q7..Q0])

" C
" L O R S S D D Q Q
" K E S 2 1 0 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0
[C,0,1,X,X,X,X,X,X,X,X,X,X] -> [0,0,0,0,0,0,0,0,0,0]; " set
[C,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1] -> [0,0,0,0,0,1,1,1,1,1]; " no shift
[C,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0] -> [1,1,1,0,0,0,0,0,0,1]; " shift 1
[C,0,0,0,0,1,0,0,0,0,0,0,1,1,1,1] -> [0,0,1,1,1,1,1,0,0,0]; " 2
[C,0,0,0,0,1,1,1,1,1,1,0,0,0,0,0] -> [1,0,0,0,0,0,1,1,1,1]; " 3
[C,0,0,0,1,0,0,0,0,0,0,0,1,1,1,1] -> [1,1,1,1,0,0,0,0,0,0]; " 4
[C,0,0,0,1,0,1,1,1,1,1,1,0,0,0,0] -> [0,0,0,1,1,1,1,1,1,0]; " 5
[C,0,0,0,1,1,0,0,0,0,0,0,1,1,1,1] -> [1,1,0,0,0,0,0,1,1,1]; " 6
[C,0,0,0,1,1,1,1,1,1,1,1,0,0,0,0] -> [0,1,1,1,1,1,0,0,0,0]; " 7

end barrel_shifter_8

```



Figure 5. Reduced Equations

ABEL(tm) Version 1.19 - Document Generator  
 8-bit Barrel Shifter in a GAL20V8 April 16, 1986 Joe Eng  
 Lattice Semiconductor  
 Equations for Module barrel\_shifter\_8  
 Device U9

## Reduced Equations:

```
Q0 := (D0 & !RS & !S0 & !S1 & !S2
# D1 & !RS & S0 & S1 & S2
# D2 & !RS & !S0 & S1 & S2
# D3 & !RS & S0 & !S1 & S2
# D4 & !RS & !S0 & !S1 & S2
# D5 & !RS & S0 & S1 & !S2
# D6 & !RS & !S0 & S1 & !S2
# D7 & !RS & S0 & !S1 & !S2);
```

```
Q1 := (D0 & !RS & S0 & !S1 & !S2
# D1 & !RS & !S0 & !S1 & !S2
# D2 & !RS & S0 & S1 & S2
# D3 & !RS & !S0 & S1 & S2
# D4 & !RS & S0 & !S1 & S2
# D5 & !RS & !S0 & !S1 & S2
# D6 & !RS & S0 & S1 & !S2
# D7 & !RS & !S0 & S1 & !S2);
```

```
Q2 := (D0 & !RS & !S0 & S1 & !S2
# D1 & !RS & S0 & !S1 & !S2
# D2 & !RS & !S0 & !S1 & !S2
# D3 & !RS & S0 & S1 & S2
# D4 & !RS & !S0 & S1 & S2
# D5 & !RS & S0 & !S1 & S2
# D6 & !RS & !S0 & !S1 & S2
# D7 & !RS & S0 & S1 & !S2);
```

```
Q3 := (D0 & !RS & S0 & S1 & !S2
# D1 & !RS & !S0 & S1 & !S2
# D2 & !RS & S0 & !S1 & !S2
# D3 & !RS & !S0 & !S1 & !S2
# D4 & !RS & S0 & S1 & S2
# D5 & !RS & !S0 & S1 & S2
# D6 & !RS & S0 & !S1 & S2
# D7 & !RS & !S0 & !S1 & S2);
```

Figure 5. (cont'd)

```

Q4 := (D0 & !RS & !S0 & !S1 & S2
# D1 & !RS & S0 & S1 & !S2
# D2 & !RS & !S0 & S1 & !S2
# D3 & !RS & S0 & !S1 & !S2
# D4 & !RS & !S0 & !S1 & !S2
# D5 & !RS & S0 & S1 & S2
# D6 & !RS & !S0 & S1 & S2

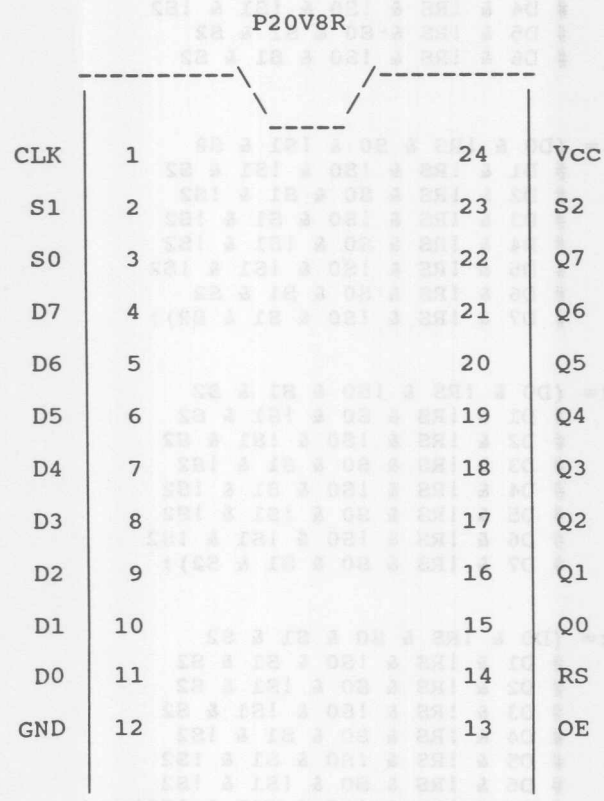
Q5 := (D0 & !RS & S0 & !S1 & S2
# D1 & !RS & !S0 & !S1 & S2
# D2 & !RS & S0 & S1 & !S2
# D3 & !RS & !S0 & S1 & !S2
# D4 & !RS & S0 & !S1 & !S2
# D5 & !RS & !S0 & !S1 & !S2
# D6 & !RS & S0 & S1 & S2
# D7 & !RS & !S0 & S1 & S2);

Q6 := (D0 & !RS & !S0 & S1 & S2
# D1 & !RS & S0 & !S1 & S2
# D2 & !RS & !S0 & !S1 & S2
# D3 & !RS & S0 & S1 & !S2
# D4 & !RS & !S0 & S1 & !S2
# D5 & !RS & S0 & !S1 & !S2
# D6 & !RS & !S0 & !S1 & !S2
# D7 & !RS & S0 & S1 & S2);

Q7 := (D0 & !RS & S0 & S1 & S2
# D1 & !RS & !S0 & S1 & S2
# D2 & !RS & S0 & !S1 & S2
# D3 & !RS & !S0 & !S1 & S2
# D4 & !RS & S0 & S1 & !S2
# D5 & !RS & !S0 & S1 & !S2
# D6 & !RS & S0 & !S1 & !S2
# D7 & !RS & !S0 & !S1 & !S2);

```

Figure 6. Pinout Diagram



### DESCRIPTION

Widely used in computer and data communications circuits, multiplexers route one of several input banks to an output, based on the condition of select inputs. This particular version has 4 input banks, each 4-bits wide (Figure 1); therefore, two select lines are required to choose 1 of 4 inputs, as shown in the function table of Figure 2. Possible applications for our multiplexer include bus selection in a multibus computer environment, or data manipulation in an arithmetic/logic circuit.

With a total of 16 multiplexer inputs and two Select inputs, this design is well suited for the GAL20V8. The pinout chosen for this example is shown in Figure 3; actual pin placement of the multiplexer outputs is not critical since the versatility of the GAL20V8 allows the designer to choose that combination of output pins that best suits the board layout. The device was programmed using ABEL; the logic design input files are shown in Figure 4, with reduced equations shown in the document-generator file of Figure 5. The 'fuse' map is shown in Figure 6. □

Figure 1. Block Diagram

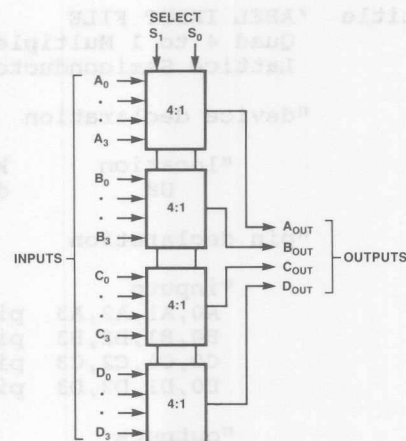
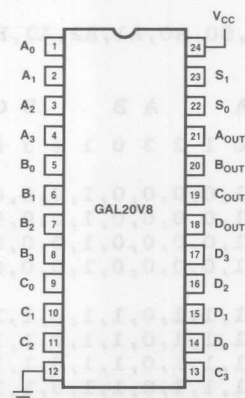


Figure 2. Function Table

S <sub>1</sub>	S <sub>0</sub>	A <sub>OUT</sub>	B <sub>OUT</sub>	C <sub>OUT</sub>	D <sub>OUT</sub>
0	0	A <sub>0</sub>	B <sub>0</sub>	C <sub>0</sub>	D <sub>0</sub>
0	1	A <sub>1</sub>	B <sub>1</sub>	C <sub>1</sub>	D <sub>1</sub>
1	0	A <sub>2</sub>	B <sub>2</sub>	C <sub>2</sub>	D <sub>2</sub>
1	1	A <sub>3</sub>	B <sub>3</sub>	C <sub>3</sub>	D <sub>3</sub>

Figure 3. Pinout Diagram



# 4:1 MULTIPLEXER

```

module quad_4to1_mux

title 'ABEL INPUT FILE
      Quad 4 to 1 Multiplexer in a GAL20V8
      Lattice Semiconductor
      April 17, 1986
      Joe Eng'

"device declaration

      "location      keyword      device code
      U8              device       'P20V8S';

"pin declaration

      "inputs
      A0,A1,A2,A3      pin 1,2,3,4;
      B0,B1,B2,B3      pin 5,6,7,8;
      C0,C1,C2,C3      pin 9,10,11,13;
      D0,D1,D2,D3      pin 14,15,16,17;

      "outputs
      Aout,Bout,Cout,Dout pin 21,20,19,18;

      "control
      S0,S1 pin 22,23;

equations

      Aout = (!S1 & !S0 & A0) # (!S1 & S0 & A1) #
              (S1 & !S0 & A2) # (S1 & S0 & A3);

      Bout = (!S1 & !S0 & B0) # (!S1 & S0 & B1) #
              (S1 & !S0 & B2) # (S1 & S0 & B3);

      Cout = (!S1 & !S0 & C0) # (!S1 & S0 & C1) #
              (S1 & !S0 & C2) # (S1 & S0 & C3);

      Dout = (!S1 & !S0 & D0) # (!S1 & S0 & D1) #
              (S1 & !S0 & D2) # (S1 & S0 & D3);

test_vectors

      ([S1,S0,A0,A1,A2,A3,B0,B1,B2,B3,C0,C1,C2,C3,D0,D1,D2,D3] ->
      [Aout,Bout,Cout,Dout])

" S S A      A B      B C      C D      D      outputs
" 1 0 0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3      A B C D

      " select
      [0,0,1,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,1] -> [1,0,0,0]; "A0,B0,C0,D0
      [0,1,1,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,1] -> [0,1,0,0]; "A1,B1,C1,D1
      [1,0,1,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,1] -> [0,0,1,0]; "A2,B2,C2,D2
      [1,1,1,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,1] -> [0,0,0,1]; "A3,B3,C3,D3

      [0,0,1,1,1,0,1,1,0,1,1,0,1,1,0,1,1,1,1] -> [1,1,1,0]; "A0,B0,C0,D0
      [0,1,1,1,1,0,1,1,0,1,1,0,1,1,0,1,1,1,1] -> [1,1,0,1]; "A1,B1,C1,D1
      [1,0,1,1,1,0,1,1,0,1,1,0,1,1,0,1,1,1,1] -> [1,0,1,1]; "A2,B2,C2,D2
      [1,1,1,1,1,0,1,1,0,1,1,0,1,1,0,1,1,1,1] -> [0,1,1,1]; "A3,B3,C3,D3

```

Figure 5. Reduced Equations

ABEL(tm) Version 1.19 - Document Generator  
 Quad 4 to 1 Multiplexer in a GAL20V8 April 17, 1986  
 Lattice Semiconductor Joe Eng  
 Equations for Module quad\_4to1\_mux

Device U8

#### Reduced Equations:

Aout = (A0 & !S0 & !S1  
 # A1 & S0 & !S1  
 # A2 & !S0 & S1  
 # A3 & S0 & S1);

Bout = (B0 & !S0 & !S1  
 # B1 & S0 & !S1  
 # B2 & !S0 & S1  
 # B3 & S0 & S1);

Cout = (C0 & !S0 & !S1  
 # C1 & S0 & !S1  
 # C2 & !S0 & S1  
 # C3 & S0 & S1);

Dout = (D0 & !S0 & !S1  
 # D1 & S0 & !S1  
 # D2 & !S0 & S1  
 # D3 & S0 & S1);





## DESCRIPTION

Encoding is the process of converting several lines of input data into a compact code. This code is then either transmitted along a system bus (usually of limited capacity) to be decoded elsewhere, or stored in memory for future use. A priority encoder functions as a normal encoder when only one input is active; that input's code appears at the output. When more than one input is active, the priority encoder selects the input with the highest priority, ignoring all others. The priority structure is set by the logic equations programmed into the GAL16V8, and the user then makes the appropriate pin assignments.

Our priority encoder (Figure 1) has a few extra features to enhance its capabilities. Both an input enable and an output enable are included to allow for independent control by the input and output circuits. Additionally, a group strobe (GS) output signal is provided to notify the output circuit when a valid output code is present. The truth table for the encoder is shown in Figure 2, and GAL16V8 pinout diagram in Figure 3.

The logic equations for this circuit are quite long. Fortunately, the CUPL programming language, which was used for this example, allows shortcuts in the logic description. As shown in the design input file (Figure 4), the equation for  $Y_0$  has been written in long form.  $Y_1$  and  $Y_2$  were considerably easier to write, thanks to the use of the 'equality operator.' The simulation file is shown in Figure 5, and CUPL provides the expanded product terms in Figure 6. The symbol table, which lays out the pin definitions and variable names in tabular form, is shown in Figure 7.

Figure 1. Block Diagram

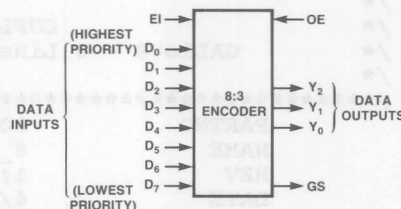
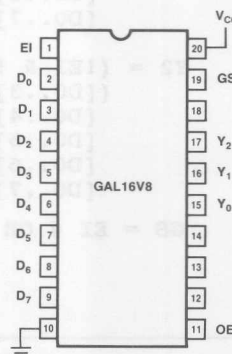


Figure 2. Function Table

EI	D <sub>0</sub>	D <sub>1</sub>	D <sub>2</sub>	D <sub>3</sub>	D <sub>4</sub>	D <sub>5</sub>	D <sub>6</sub>	D <sub>7</sub>	OE	Y <sub>2</sub>	Y <sub>1</sub>	Y <sub>0</sub>	GS
1	X	X	X	X	X	X	X	X	1	0	0	0	1
X	X	X	X	X	X	X	X	X	0	0	0	0	1
0	0	X	X	X	X	X	X	X	0	0	0	0	0
0	1	0	X	X	X	X	X	X	0	0	0	1	0
0	1	1	0	X	X	X	X	X	0	0	1	0	0
0	1	1	1	0	X	X	X	X	0	0	1	1	0
0	1	1	1	1	0	X	X	X	0	1	0	0	0
0	1	1	1	1	1	0	X	X	0	1	0	1	0
0	1	1	1	1	1	1	0	X	0	1	1	0	0
0	1	1	1	1	1	1	1	0	0	1	1	1	0
0	1	1	1	1	1	1	1	1	0	1	1	1	1

Figure 3. Pinout Diagram



```

/*****
/*
/*          CUPL INPUT FILE
/*          GAL16V8 - 8 Line to 3 Line Priority Encoder
/*
/*
*****/
PARTNO      803PLD;
NAME        8_3ENCOD;
REV         1;
DATE        4/18/86;
DESIGNER    Jerry ;
COMPANY     Lattice Semiconductor;
ASSEMBLY    Interface Board;
LOCATION     U09;

/* inputs */

pin [2..9] = [D0..7];
pin [1,11] = [EI,OE]; /* Enable Input,Output Enable */

/* outputs */

pin [15..17] = [Y0..2];
pin 19 = GS; /* Group Strobe */

/* logic equations */

Y0 = (!EI & !OE) &
      ((D0 & !D1) #
        (D0 & D1 & D2 & !D3) #
        (D0 & D1 & D2 & D3 & D4 & !D5) #
        (D0 & D1 & D2 & D3 & D4 & D5 & D6 & !D7) #
        (D0 & D1 & D2 & D3 & D4 & D5 & D6 & D7));

Y1 = (!EI & !OE) &
      ((D0 & D1 & !D2) #
        [D0..2]:& & !D3 #
        [D0..5]:& & !D6 #
        [D0..6]:& & !D7 #
        [D0..7]:&);

Y2 = (!EI & !OE) &
      ([D0..3]:& & !D4 #
        [D0..4]:& & !D5 #
        [D0..5]:& & !D6 #
        [D0..6]:& & !D7 #
        [D0..7]:&);

GS = EI # OE # [D0..7]:&;

```

Figure 5. Simulation File

```

/*****
/*
/*          CUPL SIMULATION FILE          */
/*          GAL16V8 - 8 Line to 3 Line Priority Encoder          */
/*
/*****

PARTNO      803PLD;
NAME        8_3ENCOD;
REV         1;
DATE        4/18/86;
DESIGNER     Jerry;
COMPANY      Lattice Semiconductor;
ASSEMBLY     Interface Board;
LOCATION      U09;

order:
EI,%1,OE,%2,D0,%1,D1,%1,D2,%1,D3,%1,D4,%1,D5,%1,
D6,%1,D7,%2,Y2,%1,Y1,%1,Y0,%2,GS;

vectors:
/*          D          D Y Y          */
/* EI OE 0 1 2 3 4 5 6 7  2 1 0 GS */

1 X  X X X X X X X X  L L L H /* test Enable Input */
X 1  X X X X X X X X  L L L H /* test Output Enable */
0 0  1 1 1 1 1 1 1 1  H H H H /* all inputs high */

0 0  0 0 0 0 0 0 0 0  L L L L /* test */
0 0  1 0 0 0 0 0 0 0  L L H L /* priority */
0 0  1 1 0 0 0 0 0 0  L H L L /* function */
0 0  1 1 1 0 0 0 0 0  L H H L
0 0  1 1 1 1 0 0 0 0  H L L L
0 0  1 1 1 1 1 0 0 0  H L H L
0 0  1 1 1 1 1 1 0 0  H H L L
0 0  1 1 1 1 1 1 1 0  H H H L

```

Figure 6. Expanded Product Terms

```

*****
CUPL DOCUMENTATION FILE
*****
                        8_3ENCOD
*****

CUPL      2.10B1
Device    gl6v8s  Library DLIB-d-55-8
Created   Fri Apr 18 15:52:32 1986
Name      8_3ENCOD
Partno    803PLD
Revision  1
Date      4/18/86
Designer  Jerry
Company   Lattice Semiconductor
Assembly  Interface Board
Location  U09

```

```

=====
Expanded Product Terms
=====

```

```

GS =>
    EI
    # OE
    # D0 & D1 & D2 & D3 & D4 & D5 & D6 & D7

Y0 =>
    D0 & !D1 & !EI & !OE
    # D0 & D1 & D2 & !D3 & !EI & !OE
    # D0 & D1 & D2 & D3 & D4 & !D5 & !EI & !OE
    # D0 & D1 & D2 & D3 & D4 & D5 & D6 & !EI & !OE

Y1 =>
    D0 & D1 & !D2 & !EI & !OE
    # D0 & D1 & D2 & !D3 & !EI & !OE
    # D0 & D1 & D2 & D3 & D4 & D5 & !EI & !OE

Y2 =>
    D0 & D1 & D2 & D3 & !EI & !OE

```

Figure 7. Symbol Table

Symbol Table

Pin	Variable				Pterms	Max	Min
Pol	Name	Ext	Pin	Type	Used	Pterms	Level
	D0		2	V	-	-	-
	D1		3	V	-	-	-
	D2		4	V	-	-	-
	D3		5	V	-	-	-
	D4		6	V	-	-	-
	D5		7	V	-	-	-
	D6		8	V	-	-	-
	D7		9	V	-	-	-
	EI		1	V	-	-	-
	GS		19	V	3	8	1
	OE		11	V	-	-	-
	Y0		15	V	4	8	1
	Y1		16	V	3	8	1
	Y2		17	V	1	8	1

LEGEND      F : field      D : default variable      M : extended node  
                  N : node      I : intermediate variable      T : function  
                  V : variable      X : extended variable      U : undefined



## DESCRIPTION

The GAL16V8 is used to implement a password decoder for use in systems where restricted access is critical. Extremely difficult to 'crack,' this password decoder permits access only when it encounters a correct sequence of input variables and internal states.

The device is configured as a state machine within a state machine. The first state machine, or master sequencer, verifies when an expected state is reached in the second machine, or slave sequencer. Then it increments to the next state and monitors the slave sequencer for the next expected state. The slave sequencer tests the inputs and internal states for the expected state or sequence of states.

As shown in the state diagram of Figure 1, the master sequencer is a four-state machine that increments when the proper state is reached by the slave sequencer. There are three different decoding laws (described below) used by the slave to determine this proper state for incrementing the master. When each law is satisfied in the slave, the master increments. Upon reaching state D, the asynchronous output CODE is enabled and permits access to the restricted system resource, providing the inputs to the device are in the proper state. Otherwise, an incorrect CODE output will not allow access.

Figure 1. Master Sequencer State Diagram

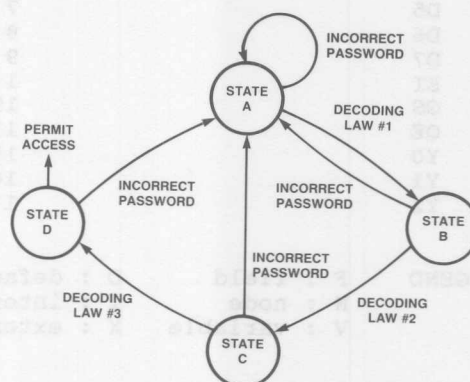
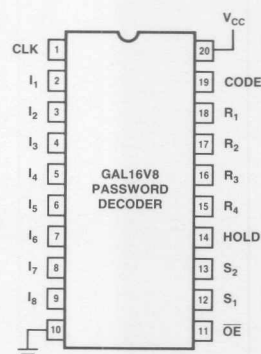


Figure 2. Pinout Diagram



The pinout diagram is shown in Figure 2. The slave sequencer monitors inputs  $I[8..1]$  and internal states  $R[4..1]$ , as well as state bits  $S[1..0]$  and  $HOLD$ . It should be noted that  $HOLD$ ,  $R[4..1]$  and  $S[1..0]$  are internal states only and not used as device outputs. The three different decoding laws that this portion of the circuit is programmed to detect are, in expected sequence:

- 1) Exclusive-Or Function
- 2) Input-dependent (inputs only)
- 3) Output-dependent (outputs only)

At power on, the decoding and state registers are automatically reset and State A is selected. Note that there is an inversion between the register and the output pin, so that when the registers are reset, the output pin actually goes high. On the rising edge of the clock, the first decoding law is checked, namely,  $R(i) : + : I(i)$ . Notice in the simulation that the master-sequencer state bits  $S[1..0]$  do not increment until  $HOLD$  is removed. This occurs once all inputs  $I[8..1]$  and all outputs  $R[4..1]$  are low. This requires two vectors, the first to force the outputs low and the next to set the inputs low. Now the state bits increment to  $[00]$ ,  $HOLD$  goes active, and the decoder is ready to check for the second law, namely, the input-dependent law that  $I[8..1] = HIGH$ .

When this condition becomes true,  $HOLD$  is removed for one clock cycle and the state bits  $S[1..0]$  increment to State C, or  $[01]$ . The third decoding law, output-dependent counting of bits  $R[4..1]$ , now becomes the requirement for the next increment of state bits  $S[1..0]$ . This law is a simple counting from  $R[4..1] = [0000]$  to  $R[4..1] = [1111]$ .

Upon reaching this condition,  $HOLD$  is again removed for one clock cycle and state bits  $S[1..0]$  increment to  $[10]$ . This causes asynchronous output  $CODE$  to be enabled, and if the necessary condition of  $I[8..1] = [10101010]$  exists,  $CODE$  will be high and the restricted path will be enabled. If that particular input condition does not exist,  $CODE$  will be low, causing access to be denied. On the next clock,  $CODE$  is disabled and the password decoder goes back to its initial state.

Once the GAL16V8 is programmed and secured, its resistance to fraudulent access is nearly perfect. Consider, for example, calculation of the number of cycles necessary to decode the device if the coding laws are known. For each state, the combination of  $I[8..1]$  and  $R[4..1]$  gives  $2^{12}$  possible values. Because each decoding law for each state is independent of those for any other state, the total number of possible combinations is  $2^{12} \times 2^{12} \times 2^{12} \times 2^{12} = 2^{48}$ . With a 50ns cycle time, it would take over 162 years to decode the GAL16V8. This does not even take into account the effect of any trap combinations which might be thrown in. Also, if the decoding laws are not known...

The decoding laws chosen here are simple ones to demonstrate the application. More complex schemes can be devised to make fraudulent access even more difficult, with the only limitation being the number of product terms per output. Also, if the same decoding laws and state transitions but a different output code is programmed into several devices in parallel, the width of the access code is easily expanded. □

**Figure 3. ABEL Input File**

```

module password
title
'ABEL INPUT FILE
Password decoder
Designer: Jerry Greiner, Lattice Semiconductor
Device: GAL16V8
4/5/86'

passkey device 'p16v8r'; Target Device: GAL16V8

" constants:

H = 1;
L = 0;
x = .X.;
c = .C.;
Z = .Z.;

" pin names:

clk,I1,I2,I3,I4,I5,I6,I7,I8 pin 1,2,3,4,5,6,7,8,9;
OE,S1,S0,HOLD,R4,R3,R2,R1,CODE pin 11,12,13,14,15,16,17,18,19;

equations

CODE = I8 & !I7 & I6 & !I5 & I4 & !I3 & I2 & !I1;

ENABLE CODE = S1 & !S0;

S0 := S0 & HOLD
# !S0 & !HOLD;

S1 := S1 & HOLD
# !S1 & S0 & !HOLD
# S1 & !S0 & !HOLD;

!HOLD := S1 & S0 & !R4 & !R3 & !R2 & !R1
& !I8 & !I7 & !I6 & !I5 & !I4 & !I3 & !I2 & !I1
# !S1 & !S0 & I1 & I2 & I3 & I4 & I5 & I6 & I7 & I8
# !S1 & S0 & R4 & R3 & R2 & R1
# S1 & !S0;

R4 := S1 & S0 & (!R4 & I4 # R4 & !I4)
# !S1 & S0 & (!R4 & R3 & R2 & R1 # R4 & !R3 # R4 & !R2 # R4 & !R1)
# S1 & S0 & HOLD & !R4 & !R3 & !R2 & !R1;

R3 := S1 & S0 & (!R3 & I3 # R3 & !I3)
# !S1 & S0 & (!R3 & R2 & R1 # R3 & !R2 # R3 & !R1);

R2 := S1 & S0 & (!R2 & I2 # R2 & !I2)
# !S1 & S0 & (!R2 & R1 # R2 & !R1);

R1 := S1 & S0 & (!R1 & I1 # R1 & !I1)
# !S1 & S0 & !R1;

```

Figure 3. (cont'd)

```
test_vectors ( [clk, I8, I7, I6, I5, I4, I3, I2, I1] ->
               [R4, R3, R2, R1, HOLD, S1, S0, CODE] )
```

		H	C	
"	C	O	O	
" L I I I I I I I I	R R R R R	L	S S D	
" K 8 7 6 5 4 3 2 1	4 3 2 1	D	1 0 E	

```

[x,x,x,x,x,x,x,x,x] -> [1,1,1,1, 1, 1,1, Z]; "initialization
[c,x,x,x,x,x,H,H,H,H] -> [0,0,0,0, 1, 1,1, Z];
[c,L,L,L,L,L,L,L,L] -> [1,0,0,0, 0, 1,1, Z]; "test first law
[c,x,x,x,x,x,x,x,x,x] -> [1,0,0,0, 1, 0,0, Z]; "increment STATE
[c,H,H,H,H,H,H,H,H,H] -> [0,0,0,0, 0, 0,0, Z]; "test second law
[c,x,x,x,x,x,x,x,x,x] -> [0,0,0,0, 1, 0,1, Z]; "increment STATE
[c,x,x,x,x,x,x,x,x,x] -> [0,0,0,1, 1, 0,1, Z]; "set up third law
[c,x,x,x,x,x,x,x,x,x] -> [0,0,1,0, 1, 0,1, Z]; "third law active
[c,x,x,x,x,x,x,x,x,x] -> [0,0,1,1, 1, 0,1, Z];
[c,x,x,x,x,x,x,x,x,x] -> [0,1,0,0, 1, 0,1, Z];
[c,x,x,x,x,x,x,x,x,x] -> [0,1,0,1, 1, 0,1, Z];
[c,x,x,x,x,x,x,x,x,x] -> [0,1,1,0, 1, 0,1, Z];
[c,x,x,x,x,x,x,x,x,x] -> [0,1,1,1, 1, 0,1, Z];
[c,x,x,x,x,x,x,x,x,x] -> [1,0,0,0, 1, 0,1, Z]; "third law
[c,x,x,x,x,x,x,x,x,x] -> [1,0,0,1, 1, 0,1, Z]; "still active
[c,x,x,x,x,x,x,x,x,x] -> [1,0,1,0, 1, 0,1, Z];
[c,x,x,x,x,x,x,x,x,x] -> [1,0,1,1, 1, 0,1, Z];
[c,x,x,x,x,x,x,x,x,x] -> [1,1,0,0, 1, 0,1, Z];
[c,x,x,x,x,x,x,x,x,x] -> [1,1,0,1, 1, 0,1, Z];
[c,x,x,x,x,x,x,x,x,x] -> [1,1,1,0, 1, 0,1, Z];
[c,x,x,x,x,x,x,x,x,x] -> [1,1,1,1, 1, 0,1, Z];
[c,x,x,x,x,x,x,x,x,x] -> [0,0,0,0, 0, 0,1, Z]; "test third law
[c,H,L,H,L,H,L,H,L] -> [0,0,0,1, 1, 1,0, 1]; "enable access
    
```

end

## DESCRIPTION

This example demonstrates a distinct advantage of the GAL family's configurable architecture. Here, an address decoder (Figure 1) for enabling up to six I/O devices in a system employs a built-in wait-state generator, to accommodate either fast or slow devices. The wait-state generator causes the READY output to go inactive for certain addresses (IOSEL1 and IOSEL4), then become active again one clock cycle later. This requires a single state bit, thus only one register is needed. Since the GAL16V8 can be configured with from 1 to 8 registers, 7 nonregistered outputs remain, allowing for a READY output plus 6 I/O-decode outputs (Figure 2). A nonregistered output was chosen for READY, so that the signal could be wire-OR'ed with other system READYS.

As a comparison, implementing this circuit with a traditional 16R4 registered PAL device (in which 4 outputs must be registered) would allow only 3 I/O-select outputs, while wasting unused registered outputs. This example also shows the advantage of the GAL16V8's power-up reset feature, which allows the state machine to be powered up in a known state without tying up an input as a reset term. This is especially important in a decoder, where a large number of address lines is often needed to get the required addressing granularity. The circuitry forces the clock input to a high TTL state during the processor's power-up reset. The CUPL design input file is shown in Figure 3, and simulation file in Figure 4.

Figure 1. Block Diagram

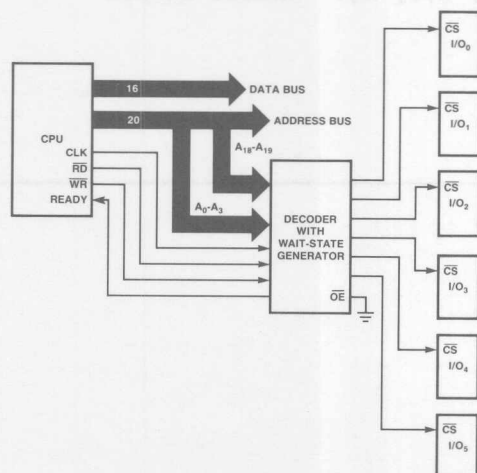


Figure 2. Pinout Diagram

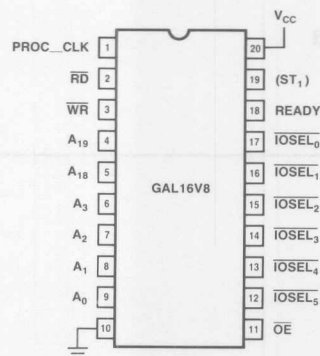


Figure 3. CUPL Input File

```

/*****
/*
/*          CUPL INPUT FILE
/*      Design input for Decoder with Wait State Generator
/*
/*
/*      Allowable Target Device Types:  GAL16V8
/*
*****/

PARTNO      DECWAIT ;
NAME        DECODER WITH WAIT STATE GENERATOR ;
DATE        10/28/85 ;
REV         01 ;
DESIGNER    Jerry Greiner;
COMPANY     Lattice Semiconductor ;
ASSEMBLY    1 ;
LOCATION     U12 ;

/** Inputs **/

PIN 1       = PROC_CLK ;
PIN [2,3]   = ![RD,WR] ;
PIN [4,5]   = [A19..18] ;
PIN [6..9]  = [A3..0] ;
PIN 11      = !OE ;

/** Outputs **/

PIN 19      = ST1 ;
PIN 18      = READY ;
PIN 17      = !IOSEL0;
PIN 16      = !IOSEL1 ;
PIN 15      = !IOSEL2 ;
PIN 14      = !IOSEL3 ;
PIN 13      = !IOSEL4 ;
PIN 12      = !IOSEL5 ;

/** Declarations and Intermediate Variable Definitions **/

IOSEG = A19 & !A18 ; /* Decode selects this chip */
SEL0 = !A3 & !A2 & !A1 & !A0 ; /* Output selection encoded by */
SEL1 = !A3 & !A2 & !A1 & A0 ; /* intermediate variables */
SEL2 = !A3 & !A2 & A1 & !A0 ;
SEL3 = !A3 & !A2 & A1 & A0 ;
SEL4 = !A3 & A2 & !A1 & !A0 ;
SEL5 = !A3 & A2 & !A1 & A0 ;
ACC = RD # WR ; /* ACCess whether READ or WRITE */
IOSEL0 = ACC & IOSEG & SEL0 ;
IOSEL1 = ACC & IOSEG & SEL1 ;
IOSEL2 = ACC & IOSEG & SEL2 ;
IOSEL3 = ACC & IOSEG & SEL3 ;
IOSEL4 = ACC & IOSEG & SEL4 ;
IOSEL5 = ACC & IOSEG & SEL5 ;
ST1.D = ACC & IOSEG & (SEL1 # SEL4) & !ST1 ; /* Wait one cycle */
READY = RD & !RD ;
READY.OE = ST1 ; /* Disable output for one clock */

```



Figure 4. CUPL Simulation File

```

/*****
/*
/*          CUPL SIMULATION FILE
/*      Simulation for Decoder with Wait State Generator
/*
/*
/* *****
/* Allowable Target Device Types:  GAL16V8
/* *****
PARTNO      DECAWAIT ;
NAME        DECODER WITH WAIT STATE GENERATOR;
DATE        10/28/85 ;
REV         01 ;
DESIGNER    JERRY GREINER;
COMPANY     LATTICE SEMICONDUCTOR;
ASSEMBLY    1 ;
LOCATION     U12 ;

ORDER:
PROC CLK, %2, !RD, %2, !WR, %2, A19,A18, %2, A3,A2,
A1,A0, %2, !oe, %4, ST1, %2, READY, %2, !IOSEL0,!IOSEL1,!IOSEL2,
!IOSEL3,!IOSEL4,!IOSEL5;
VECTORS:
$msg "          !!!!! " ;
$msg "          IIIIII " ;
$msg "          R  OOOOOO " ;
$msg "          E  SSSSSS " ;
$msg " C  !  !  AA      !  S  A  EEEEEEE " ;
$msg " L  R  W  11  AAAA  O  T  D  LLLLLL " ;
$msg " K  D  R  98  3210  E  1  Y  012345 " ;
$msg " ----- " ;

O  1  1  00  0000  0  X  X  HHHHHH /* Initialize */
C  1  1  00  0000  0  L  Z  HHHHHH
O  1  1  00  0000  0  L  Z  HHHHHH
O  0  1  01  0001  0  L  Z  HHHHHH /* No operation: A19 */
C  0  1  01  0001  0  L  Z  HHHHHH /* and A18 do not */
O  1  1  01  0001  0  L  Z  HHHHHH /* decode properly */
O  1  1  01  0000  0  L  Z  HHHHHH
O  0  1  10  0001  0  L  Z  HLHHHH /* READ active, test */
C  0  1  10  0001  0  H  L  HLHHHH /* for Wait State */
O  0  1  10  0001  0  H  L  HLHHHH /* !IOSEL1 selected */
C  0  1  10  0001  0  L  Z  HLHHHH
O  1  1  00  0000  0  L  Z  HHHHHH
O  0  1  10  0100  0  L  Z  HHHHLH /* READ active, test */
C  0  1  10  0100  0  H  L  HHHHLH /* for Wait State */
O  0  1  10  0100  0  H  L  HHHHLH /* !IOSEL4 selected */
C  0  1  10  0100  0  L  Z  HHHHLH
O  1  1  00  0000  0  L  Z  HHHHHH
O  1  0  10  0000  0  L  Z  LHHHHH /* No Wait, !IOSEL0 */
O  1  0  10  0010  0  L  Z  HHLHHH /* No Wait, !IOSEL2 */
O  1  0  10  0011  0  L  Z  HHHLHH /* No Wait, !IOSEL3 */
O  1  0  10  0101  0  L  Z  HHHHHL /* No Wait, !IOSEL5 */

```

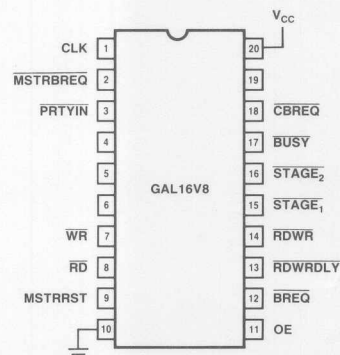
## DESCRIPTION

Bus-oriented systems offer many advantages to designers because of their inherent flexibility and expandability. Such systems require control logic to determine how each device will interact with others on the bus. This application details one scheme of so-called arbitration control. Here, a GAL16V8 (Figure 1) functions as a bus arbiter. The timing diagram is shown in Figure 2, and ABEL input file in Figure 3.

The application assumes 'daisy-chain' prioritization, which means that the device closest to the processor has the highest priority, while the device furthest from the processor has the lowest priority. The arbitration process begins with a Master Bus Request (MSTRBREQ), which is then passed through two stages (STAGE<sub>1</sub>, STAGE<sub>2</sub>) for synchronization to verify that the Request is valid and not just extraneous noise in the system. Once STAGE<sub>2</sub> is valid, lower-priority devices in the system are informed that a request has come in. The actual output vehicle of a valid STAGE<sub>2</sub> is the bus request (BREQ), which is asserted once a read or write operation begins. Next, the arbiter checks to see if the bus is already busy, in which case the arbiter waits until the bus is free. If the bus is not busy, or becomes free after waiting, the arbiter then grants control of the bus to the requesting master. At this point, BUSY then becomes active, signalling to all master devices (regardless of priority) that the bus is occupied.

Bus master status is granted by asserting read/write (RDWR), indicating to the master device to begin its read or write operation. RDWR is delayed for one clock cycle through read/write delay (RDWRDLY), to allow the bus to stabilize before the master device begins its operation. Master reset (MSTRRST) is included to allow the arbitration control to be initialized at any time. □

Figure 1. Pinout Diagram



# BUS ARBITER

Figure 2. Timing Diagram

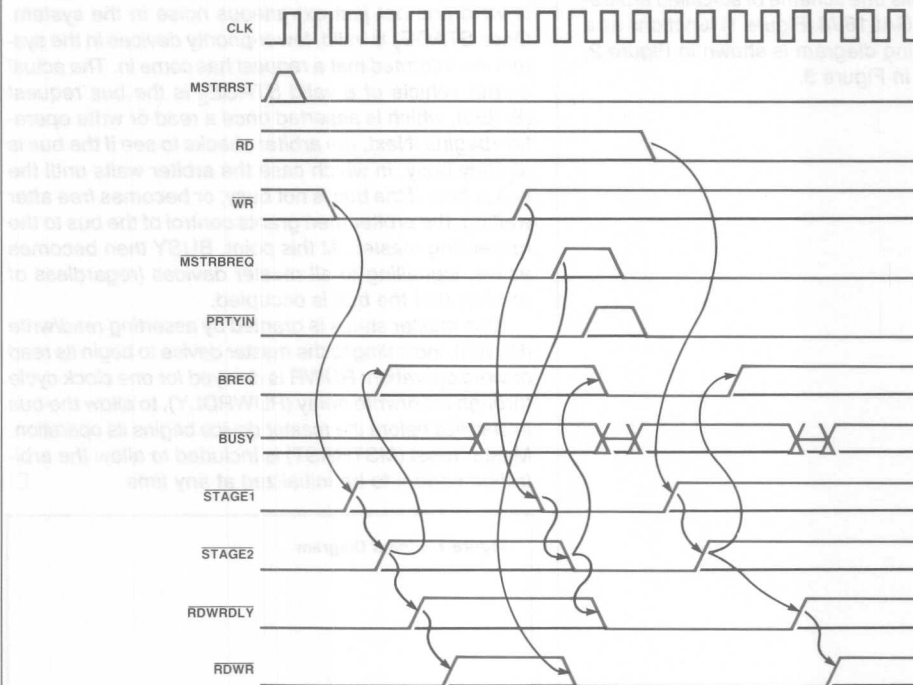


Figure 3. ABEL Design Input File

```

module    BUS_ARBITRATION

title     ' ABEL INPUT FILE
          Multibus Arbitration Control
          Jerry Greiner
          Lattice Semiconductor
          Beaverton, OR
          12/17/85'

          BUSARB device 'pl6v8r';

"Constants:

          H = 1;
          L = 0;
          X = .X.;
          C = .C.;

"Pin Names:

CLK       pin 1;
_MSTRBREQ pin 2; "MASTER BUS REQUEST IN
_PRTYIN   pin 3; "BUS PRIORITY INPUT ASSIGNED TO THIS DEVICE
_WR       pin 7; "WRITE OPERATION
_RD       pin 8; "READ OPERATION
MSTRRST   pin 9; "MASTER RESET
OE        pin 11;
_BREQ     pin 12; "BUS REQUEST OUTPUT TO ENTIRE SYSTEM
_RDWRDLY  pin 13; "DELAYS RD/WR TO ALLOW BUS TO STABILIZE
_RDWR     pin 14; "RD/WR OUTPUT SIGNAL TO SLAVE DEVICE
_STAGE1   pin 15; "FIRST STAGE SYNCHRONIZER OF 'MSTRBREQ'
_STAGE2   pin 16; "SECOND STAGE OF 'MSTRBREQ' VALIDATION
_BUSY     pin 17; "INDICATES WHETHER BUS IS BUSY
_CBREQ    pin 18;

MSTRBREQ, PRTYIN, WR, RD = !_MSTRBREQ, !_PRTYIN, !_WR, !_RD;
BREQ, RDWRDLY, RDWR, STAGE2, STAGE1, BUSY, CBREQ =
    !_BREQ, !_RDWRDLY, !_RDWR, !_STAGE2, !_STAGE1, !_BUSY, !_CBREQ;

equations

CBREQ = STAGE2 & !RDWR;

    ENABLE CBREQ = STAGE2 & !RDWR;

BREQ = STAGE2 # RDWR;

BUSY = RDWR;

    ENABLE BUSY = RDWR;

```

Figure 3. (cont'd)

```

RDWRDLY := !MSTRRST & RDWR & STAGE2 * WR
# !MSTRRST & RDWR & STAGE2 & RD
# !MSTRRST & STAGE2 & PRTYIN & !BUSY
# !MSTRRST & RDWR & PRTYIN & !CBREQ;

RDWR := !MSTRRST & MSTRBREQ & RDWR;

STAGE1 := !MSTRRST & MSTRBREQ & WR
# !MSTRRST & MSTRBREQ & RD;

STAGE2 := !MSTRRST & STAGE1;

test_vectors ( [CLK, OE, MSTRRST, !RD, !WR, !MSTRBREQ, !PRTYIN] ->
               [CBREQ, BREQ, BUSY, STAGE1, STAGE2, RDWRDLY, RDWR])

"
"      !
"      M !
"      S P S S D
"      T R C T T W
"      T B T B B B A A R R
" C R ! ! R Y R R U G G D D
" L O S R W E I E E S E E L W
" K E T D R Q N Q Q Y 1 2 Y R
"
[ C,L,H, X,X, X,X] -> [Z,0,Z,0,0,0,0]; "INITIALIZE
[ C,L,L, H,H, L,L] -> [Z,0,Z,0,0,0,0];
[ C,L,L, H,L, L,L] -> [Z,0,Z,1,0,0,0]; "BEGIN WRITE OPERATION
[ C,L,L, H,L, L,L] -> [1,1,Z,1,1,0,0]; "VERIFY VALID BUS REQUEST
[ C,L,L, H,L, L,L] -> [1,1,X,1,1,1,0]; " BY SYNCHRONIZING (STAGE1,
[ C,L,L, H,L, L,L] -> [1,1,Z,1,1,1,1]; " STAGE2)
[ C,L,L, H,L, L,L] -> [Z,1,1,1,1,1,1];
[ C,L,L, H,L, L,L] -> [Z,1,1,1,1,1,1];
[ C,L,L, H,H, L,L] -> [Z,1,1,0,1,1,1]; "END WRITE OPERATION
[ C,L,L, H,H, H,L] -> [Z,1,1,0,0,1,0];
[ C,L,L, H,H, H,H] -> [Z,0,Z,0,0,0,0]; "REMOVE FROM BUS
[ C,L,L, H,H, L,L] -> [Z,0,1,0,0,0,0];
[ C,L,L, L,H, L,L] -> [Z,0,1,1,0,0,0]; "BEGIN READ OPERATION
[ C,L,L, L,H, L,L] -> [1,1,1,1,1,0,0]; "VERIFY BUS REQUEST
[ C,L,L, L,H, L,L] -> [1,1,1,1,1,0,0]; "BUS IS BUSY, WAIT FOR
[ C,L,L, L,H, L,L] -> [1,1,1,1,1,0,0]; " GRANT.
[ C,L,L, L,H, L,L] -> [Z,1,X,1,1,1,0]; "BUS IS FREE, GET BUS AND
[ C,L,L, L,H, L,L] -> [Z,1,1,1,1,1,1]; " TELL REST OF SYSTEM THAT
[ C,L,L, L,H, L,L] -> [Z,1,1,1,1,1,1]; " BUS IS TAKEN
[ C,L,L, L,H, L,L] -> [Z,1,1,1,1,1,1];
[ C,L,L, L,H, L,L] -> [Z,1,1,1,1,1,1];
[ C,L,L, H,H, L,L] -> [Z,1,1,0,1,1,1]; "END READ OPERATION
[ C,L,L, H,H, H,L] -> [Z,1,1,0,0,1,0];

end BUS_ARBITRATION

```

### DESCRIPTION

In this application, the GAL20V8 adds two 4-bit numbers,  $A[3..0]$  and  $B[3..0]$ , plus a carry-in bit ( $C_{in}$ ), and provides a registered 4-bit result,  $S[3..0]$ , and an asynchronous carry-out ( $C_{out}$ ) at the outputs. The adder conforms to the simple rules:

$$S_n = A_n \oplus B_n \oplus C_n$$

$$C_{n+1} = A_n * B_n + (A_n + B_n) * C_n$$

Intermediate carry bits  $C[3..1]$  are required to evaluate the sum bits as indicated in the above equations. A reset input (Figure 1) is also provided to clear a sum stored in the registers.

The use of carry-in ( $C_{in}$ ) and carry-out ( $C_{out}$ ) makes the adder fully cascadable, allowing the width of the addition to be tailored to any application. The ABEL design input file is shown in Figure 2. □

Figure 1. Pinout Diagram

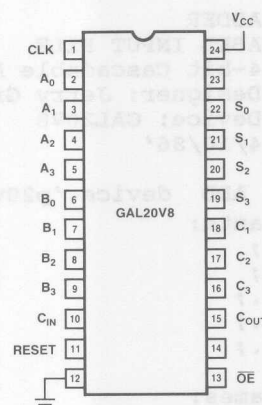




Figure 2. ABEL Design Input File

```

module ADDER
title 'ABEL INPUT FILE
      4-bit Cascadable Adder
      Designer: Jerry Greiner, Lattice Semiconductor
      Device: GAL20V8
      4/10/86'

      ADD device 'p20v8r';

" constants:
H = 1;
L = 0;
x = .X.;
c = .C.;
Z = .Z.;

" pin names:
CLK,A0,A1,A2,A3,B0,B1,B2,B3,Cin    pin 1,2,3,4,5,6,7,8,9,10;
OE,Cout,C3,C2,C1,S3,S2,S1,S0       pin 13,15,16,17,18,19,20,21,22;
RESET                               pin 11;

equations

C1 = A0 & B0 # (A0 # B0) & Cin;    "Intermediate carry
C2 = A1 & B1 # (A1 # B1) & C1;    "Intermediate carry
C3 = A2 & B2 # (A2 # B2) & C2;    "Intermediate carry
Cout = A3 & B3 # (A3 # B3) & C3;  "Carry out from sum

S0 := (A0 $ B0 $ Cin) & !RESET;   "LSB of 4-bit sum
S1 := (A1 $ B1 $ C1) & !RESET;
S2 := (A2 $ B2 $ C2) & !RESET;
S3 := (A3 $ B3 $ C3) & !RESET;    "MSB of 4-bit sum

test_vectors ([CLK,RESET,A3,A2,A1,A0,B3,B2,B1,B0,Cin] ->
              [C3,C2,C1,S3,S2,S1,S0,Cout])

"      R
"      E
" C S C
" L E A A A A B B B B i C C C S S S S u
" K T 3 2 1 0 3 2 1 0 n 3 2 1 3 2 1 0 t

[c, H, x,x,x,x, x,x,x,x, x] -> [x,x,x, 0,0,0,0, x]; "RESET SUM
[c, L, L,L,L,L, L,L,L,L, L] -> [0,0,0, 0,0,0,0, 0]; "0+0=0
[c, L, L,L,L,L, H,H,H,H, L] -> [0,0,0, 1,1,1,1, 0]; "0+15=15
[c, L, H,H,H,H, L,L,L,L, L] -> [0,0,0, 1,1,1,1, 0]; "15+0=15
[c, L, H,H,H,H, L,L,L,L, H] -> [1,1,1, 0,0,0,0, 1]; "15+0+1=0, Cout
[c, L, L,H,L,H, H,L,H,L, L] -> [0,0,0, 1,1,1,1, 0]; "5+10=15
[c, L, L,L,H,H, L,L,H,H, L] -> [0,1,1, 0,1,1,0, 0]; "3+3=6
[c, L, H,L,H,L, H,L,H,L, H] -> [0,1,0, 0,1,0,1, 1]; "10+10+1=5, Cout
[c, L, H,H,L,H, L,H,H,H, H] -> [1,1,1, 0,1,0,1, 1]; "13+7+1=5, Cout
[c, L, L,L,H,H, L,H,H,L, L] -> [1,1,0, 1,0,0,1, 0]; "3+6=9

end ADDER

```

## DESCRIPTION

This interface for a microcontroller to peripheral devices includes a 'clock-stretcher' to lengthen the allowed memory access time for slow, memory-mapped peripherals (such as analog-to-digital converters), and provides decoding circuitry for five peripherals. The address ranges that require clock-stretching can be varied according to GAL16V8 programming.

As shown in Figure 1, the GAL16V8 is used as a state sequencer with three states. An intermediate variable definition named EXTND is responsible for stretching the clock during a slow access. In this application, the slow peripheral is decoded from the address lines as  $\bar{A}_{15} \cdot \bar{A}_{14} \cdot \bar{A}_{13}$ . If the EXTND signal is not asserted, the GAL16V8 will cycle between states (01) and (00). If the EXTND signal is asserted, the GAL16V8 will cycle to state (10) and wait until a READY signal is received from the peripheral device. When READY is received, the machine returns to state (01) and assumes normal operation.

The pinout diagram is shown in Figure 2. This type of circuit requires a high-speed clock running at twice the required clock speed for the microcontroller. For a 12MHz microcontroller, the clock must run at 24MHz — still well within the speed range of the GAL16V8. The CUPL design input file is shown in Figure 3, and the simulation output in Figure 4. □

Figure 1. State Diagram

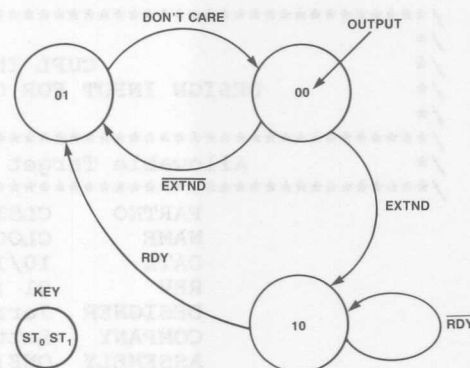


Figure 2. Pinout Diagram

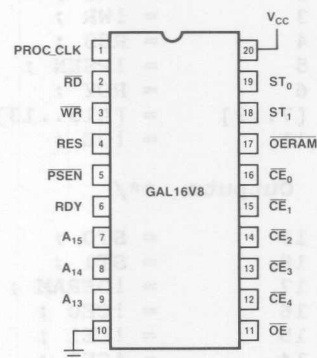


Figure 3. CUPL Design Input File

```

/*****
/*
/*          CUPL INPUT FILE          */
/*          DESIGN INPUT FOR CLOCK STRETCHER CIRCUIT          */
/*
/*          Allowable Target Device Types: GAL16V8          */
*****/

PARTNO      CLST1 ;
NAME        CLOCK STRETCHER ;
DATE        10/16/85 ;
REV         01 ;
DESIGNER    Jerry Greiner;
COMPANY     Lattice Semiconductor;
ASSEMBLY    ONE;
LOCATION     U15;

/** Inputs **/

PIN 1       = PROC_CLK ;
PIN 2       = !RD ;
PIN 3       = !WR ;
PIN 4       = RES ;
PIN 5       = !PSEN ;
PIN 6       = RDY ;
PIN [7..9]  = [A15..13] ;
PIN 11      = !OE ;

/** Outputs **/

PIN 19      = ST0 ;
PIN 18      = ST1 ;
PIN 17      = !OERAM ;
PIN 16      = !CE0 ;
PIN 15      = !CE1 ;
PIN 14      = !CE2 ;
PIN 13      = !CE3 ;
PIN 12      = !CE4 ;

/** Declarations and Intermediate Variable Definitions **/

EXTND = (RD # WR) & !A15 & !A14 & !A13 ;
ENCE = (RD # WR # PSEN) ;
OERAM = RD # PSEN ;
CE0 = ENCE & !A15 & !A14 & !A13 ;
CE1 = ENCE & !A15 & !A14 & A13 ;
CE2 = ENCE & !A15 & A14 & !A13 ;
CE3 = ENCE & !A15 & A14 & A13 ;
CE4 = ENCE & A15 & !A14 & !A13 ;
ST0.D = (!ST1 & !ST0 & EXTND & !RES) # (!ST1 & ST0 & !RDY & !RES) ;
ST1.D = (!ST1 & !ST0 & !RES & !EXTND) # (!ST1 & ST0 & RDY & !RES) ;

```

Figure 4. CUPL Simulation File

```

/*****
/*
/*          CUPL SIMULATION FILE
/*          Simulation for Clock Stretcher
/*
/*
/*          Allowable Target Device Types: GAL16V8
*****/

PARTNO    CLST1 ;
NAME      CLOCK STRETCHER ;
DATE      10/16/85 ;
REV       01 ;
DESIGNER   Jerry Greiner;
COMPANY    Lattice Semiconductor;
ASSEMBLY   ONE;
LOCATION    U15;

ORDER:
PROC CLK,%2,RES,%2,!RD,!WR,!PSEN,RDY,%2,A15,A14,A13,%2,!OE,
%4,ST0,ST1,%2,!OERAM,%2,!CE0,!CE1,!CE2,!CE3,!CE4 ;

VECTORS:
$msg "
$msg "          !          !          ";
$msg "          P          O          ";
$msg "          E          !!!!! ";
$msg " C R !!SR AAA !          R CCCCC ";
$msg " L E RWED 111 O          SS A EEEEE ";
$msg " K S DRNY 543 E          01 M 01234 ";
$msg "-----";

O 1 1110 000 0 XX H HHHHH /* Initialization. */
C 1 1110 000 0 LL H HHHHH /* Toggle between 00, */
O 0 1110 000 0 LL H HHHHH /* 01 for normal mode */
C 0 1110 000 0 LH H HHHHH
O 0 1110 000 0 LH H HHHHH
C 0 1110 000 0 LL H HHHHH
O 0 1110 000 0 LL H HHHHH
C 0 1110 000 0 LH H HHHHH
O 0 0110 000 0 LH L LHHHH /* READ active, Clock */
C 0 0110 000 0 LL L LHHHH /* Stretch required */
O 0 0110 000 0 LL L LHHHH
C 0 0110 000 0 HL L LHHHH /* Wait in state 10 */
O 0 0110 000 0 HL L LHHHH /* until RDY */
C 0 0110 000 0 HL L LHHHH

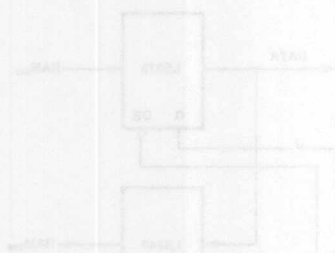
```

Figure 4. (cont'd)

0	0	0111	000	0	HL	L	LHHHH	/* RDY active, resume */
C	0	0111	000	0	LH	L	LHHHH	/* normal operation */
0	0	0110	000	0	LH	L	LHHHH	
C	0	0110	000	0	LL	L	LHHHH	
0	0	1110	000	0	LL	H	HHHHH	
C	0	1110	000	0	LH	H	HHHHH	
0	0	1010	001	0	LH	H	HLHHH	/* WRITE (!WR) active */
C	0	1010	001	0	LL	H	HLHHH	/* No Clock Stretch */
0	0	1010	001	0	LL	H	HLHHH	/* required this cycle*/
C	0	1010	001	0	LH	H	HLHHH	
0	0	1010	001	0	LH	H	HLHHH	
C	0	1010	001	0	LL	H	HLHHH	
0	0	1110	000	0	LL	H	HHHHH	/* !WR inactive, */
C	0	1110	000	0	LH	H	HHHHH	/* resume normal mode */

### DESCRIPTION

As an example of the speed and architectural flexibility of the GAL16V8, a dual-port, dynamic-RAM controller capable of controlling four banks of DRAMs is implemented. The design, whose block diagram is shown in Figure 1 and state diagrams in Figures 2 and 3, requires two GAL16V8 devices. The CUPL input listings for each device are shown in Figures 4 and 6, with respective simulation files shown in Figures 5 and 7.



The first device, the Controller, is primarily responsible for maintaining the state of the entire circuit. As shown by its state diagram in Figure 2, the Controller normally resides in the IDLE state. It can cycle to any of the states: RFGT (Refresh Grant), RQGTB (Request Grant B), or RQGTB (Request Grant B), depending on the inputs: REFRQ (Refresh Request), MRQA (Memory Request A), or MRQB (Memory Request B).

REFRQ has top priority, since the refresh cycle is of vital importance for DRAM memory retention. MRQA is arbitrarily chosen to have priority over MRQB to avoid bus contention with contiguous requests. Every REQUEST, whether a refresh request or a memory request, must receive an ACK (acknowledge) signal before the Controller will continue to cycle. Once an ACK is received, the Controller will either return to the IDLE state or perform a refresh (if REFRQ is present), and then return to the IDLE state. Cycling between RQGTB and RQGTB is also possible.

The CUPL input file for the Controller, shown in Figure 4, distinguishes output declarations from intermediate variable definitions, which greatly reduce the complexity of declarations. BK<sub>3</sub>-BK<sub>0</sub> are intermediate definitions decoded from address lines A<sub>17</sub> and A<sub>16</sub> to determine which bank will be selected. RQGTAS, RQGTBS, and RFGTS are also intermediate definitions



of Controller state paths. These are used to simplify the final output declarations.

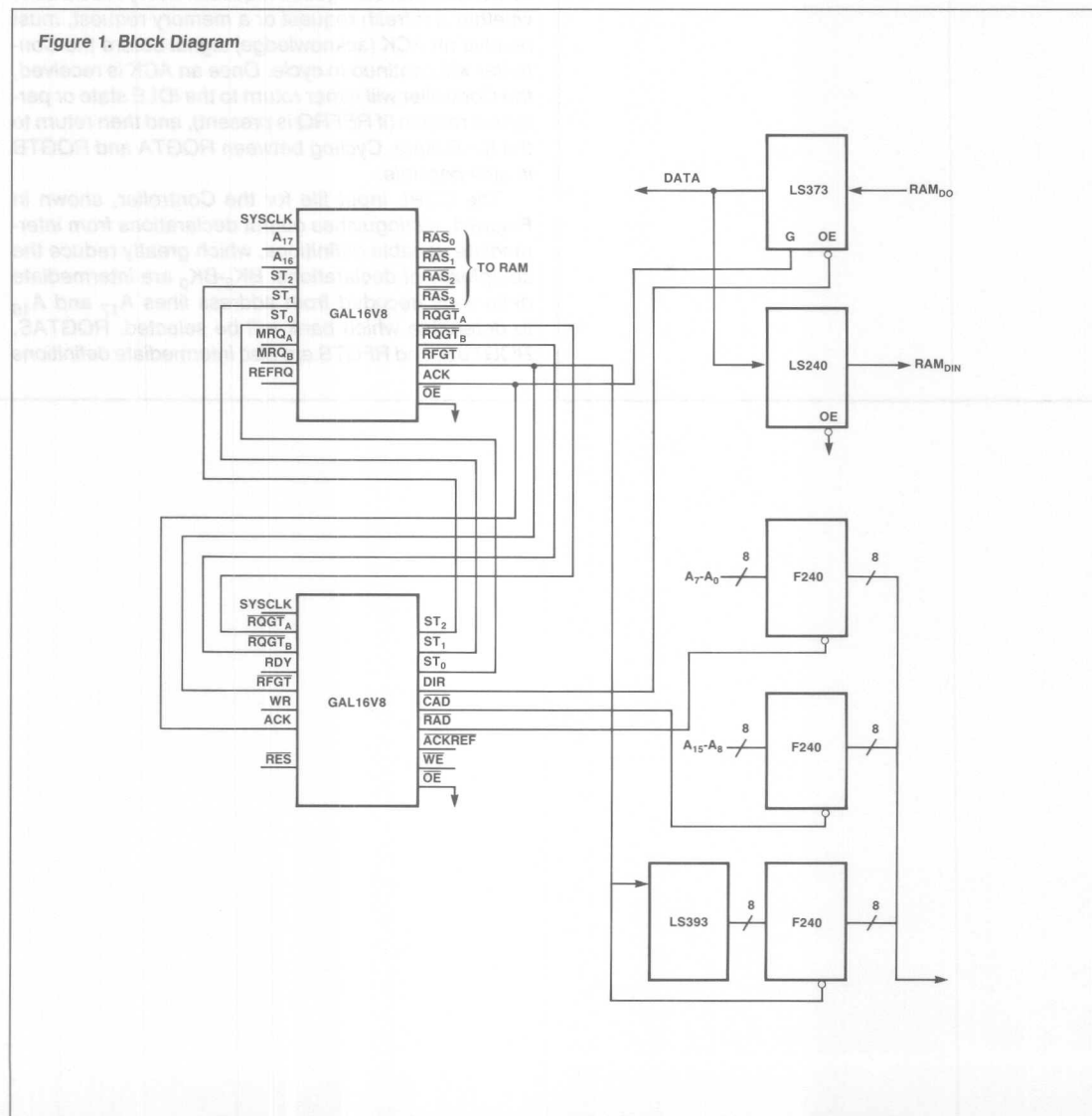
Output declarations for RQGT<sub>A</sub>, RQGT<sub>B</sub>, and RFTG are formulated by simply documenting each set of input conditions that causes the Controller to enter each state. ACK is a signal asserted by inputs the Controller receives that acknowledge the end of a memory access.

The second GAL16V8 device, called the Sequencer, is a state counter that asserts the control signals com-

municating with the DRAM section. Among these signals are: RAD (Row-Address-Data enable), CAD (Column-Address-Data enable), RAS (Row-Address Strobe), CAS (Column-Address Strobe), and ACK (Acknowledge). These signals are asserted when the Sequencer enters the proper state, as shown in the state diagram of Figure 3.

The CUPL input listing for the Sequencer is shown in Figure 6. Again, intermediate variable definitions are

Figure 1. Block Diagram



used to simplify output declarations.  $DST_8$ - $DST_1$  are intermediate definitions that name the states as decoded by the variables  $ST_2$ ,  $ST_1$ ,  $ST_0$ . Notice that a grey-code scheme, which minimizes the number of product terms, was used for the counting operation.

Next,  $ST_2$ ,  $ST_1$ , and  $ST_0$  are declared by identifying which previous states will cause each next state. For ex-

ample, to cycle from state 2 ( $DST_2$ ) to state 3 ( $DST_3$ ), variables  $ST_2$  and  $ST_1$  will be logic ones and variable  $ST_0$  will be a logic zero upon reaching the new state. This can easily be extracted from the CUPL listing. Outputs RAD and CAD are also declared using the intermediate definitions  $DST_8$ - $DST_1$ .

Figure 2. State Diagram for Controller Section

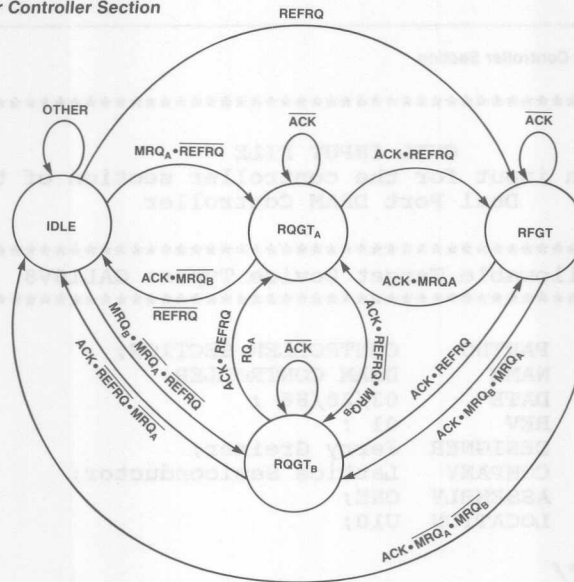
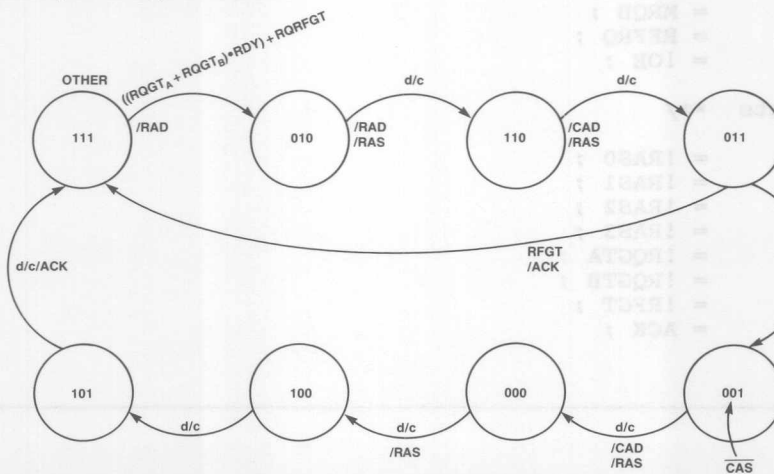


Figure 3. State Diagram for Sequencer Section



The two GAL16V8-25 devices can be clocked at cycle times as fast as 35 ns (28.5 MHz), ample enough for the tight timings required to run a DRAM at its specified access times. The GAL16V8's power-up reset feature comes in handy in this circuit, since no inputs were available for a reset term. To test the functionality of this circuit, the simulation facilities of CUPL were used.

It should be noted that although the Controller uses all eight registers in the device, the Sequencer requires seven registers and one combinational output. While the Controller could be implemented in a traditional PAL configuration (16R8), the Sequencer requires a nonstandard architecture which can only be implemented in a GAL16V8 device. This is one of the biggest advantages of GAL devices — the flexibility of the architecture. □

Figure 4. Design Input File for Controller Section

```

/*****
/*
/*          CUPL INPUT FILE
/*          Design input for the controller section of the
/*          Dual Port DRAM Controller
/*
/*
/*          Allowable Target Device Types: GAL16V8
*****/

PARTNO      CONTROLLER SECTION;
NAME        DRAM CONTROLLER;
DATE        03/28/86 ;
REV         01 ;
DESIGNER    Jerry Greiner;
COMPANY     Lattice Semiconductor;
ASSEMBLY    ONE;
LOCATION     U10;

/** Inputs **/

PIN 1       = SYCLK;
PIN [2,3]   = [A16,A17] ;
PIN [4..6]  = [ST2,ST1,ST0] ;
PIN 7       = MRQA ;
PIN 8       = MRQB ;
PIN 9       = REFRQ ;
PIN 11      = !OE ;

/** Outputs **/

PIN 19      = !RAS0 ;
PIN 18      = !RAS1 ;
PIN 17      = !RAS2 ;
PIN 16      = !RAS3 ;
PIN 15      = !RQGTB ;
PIN 14      = !RQGTB ;
PIN 13      = !RFGT ;
PIN 12      = ACK ;

```

Figure 4. (cont'd)

```

/** Declarations and Intermediate Variable Definitions */

BK0 = (!A17 & !A16) # RFGT ;
BK1 = (!A17 & A16) # RFGT ;
BK2 = (A17 & !A16) # RFGT ;
BK3 = (A17 & A16) # RFGT ;

RASEN = !ST2 & ST1 & !ST0 # ST2 & ST1 & !ST0 # !ST2 & ST1 & ST0 #
        !ST2 & !ST1 & ST0 # !ST2 & !ST1 & !ST0 ;

RAS0.D = BK0 & RASEN ;
RAS1.D = BK1 & RASEN ;
RAS2.D = BK2 & RASEN ;
RAS3.D = BK3 & RASEN ;

RQGTAS = RQGTAS & !RQGTB & !RFGT ;
RQGTBS = !RQGTAS & RQGTB & !RFGT ;
RFGTS = !RQGTAS & !RQGTB & RFGT ;
IDLE = !RQGTAS & !RQGTB & !RFGT ;

RQGTAS.D = (IDLE & MRQA & !REFRQ # RQGTAS & !ACK # RQGTBS & ACK &
            MRQA & !REFRQ # RFGTS & ACK & MRQA) & !(ACK & !MRQA &
            !MRQB & !REFRQ) ;

RQGTBS.D = (IDLE & !MRQA & !REFRQ & MRQB # RQGTBS & !ACK # RQGTAS &
            ACK & MRQB & !REFRQ # RFGTS & ACK & !MRQA & MRQB) & !(ACK &
            !MRQA & !MRQB & !REFRQ) ;

RFGT.D = (IDLE & REFRQ # RFGTS & !ACK # RQGTAS & ACK & REFRQ #
            RQGTBS & ACK & REFRQ) & !(ACK & !MRQA & !MRQB & !REFRQ) ;

ACK.D = ST2 & !ST1 & ST0 # !ST2 & ST1 & ST0 & RFGT ;

```

Figure 5. Simulation File for Controller Section

```

/*****
/*
/*          CUPL SIMULATION FILE
/*      Simulation input for the controller section of the
/*      Dual Port DRAM Controller
/*
/*
/*      Allowable Target Device Types: GAL16V8
/*
*****/

```

```

PARTNO      CONTROLLER SECTION;
NAME        DRAM CONTROLLER;
DATE        03/28/86 ;
REV         01 ;
DESIGNER    Jerry Greiner ;
COMPANY     Lattice Semiconductor ;
ASSEMBLY    ONE;
LOCATION     U10;

```

## ORDER:

```

SYSCLK,%2,A17,A16,%2,ST2,ST1,ST0,%2,MRQA,MRQB,REFRQ,%2,!OE,%4,
!RAS0,!RAS1,!RAS2,!RAS3,%2,!RQGTa,!RQGTb,!RFGT,%2,ACK;

```

Figure 5. (cont'd)

## VECTORS:

```

$msg" S          !!      ";
$msg" Y          R      !!!! RR!  ";
$msg" S          MME     RRRR  QQR  ";
$msg" C  AA  SSS  RRF  !  AAAA  GGF  A  ";
$msg" L  11  TTT  QQR  O  SSSS  TTG  C  ";
$msg" K  76  210  ABQ  E  0123  ABT  K  ";
$msg" -----

```

```

O  00  101  000  0  XXXX  XXX  X
C  00  101  000  0  HHHH  XXX  H
C  00  111  000  0  HHHH  HHH  L
C  00  111  000  0  HHHH  HHH  L
C  00  111  100  0  HHHH  LHH  L
C  00  010  100  0  LHHH  LHH  L
C  00  110  100  0  LHHH  LHH  L
C  00  011  100  0  LHHH  LHH  L
C  00  001  100  0  LHHH  LHH  L
C  00  000  100  0  LHHH  LHH  L
C  00  100  100  0  HHHH  LHH  L
C  00  101  110  0  HHHH  LHH  H
C  00  111  110  0  HHHH  HLH  L
C  11  111  010  0  HHHH  HLH  L
C  11  111  010  0  HHHH  HLH  L
C  11  010  010  0  HHHL  HLH  L
C  11  110  010  0  HHHL  HLH  L
C  11  011  010  0  HHHL  HLH  L
C  11  001  010  0  HHHL  HLH  L
C  11  000  000  0  HHHL  HLH  L
C  11  100  101  0  HHHH  HLH  L
C  11  101  101  0  HHHH  HLH  H
C  00  111  101  0  HHHH  HHL  L
C  00  111  101  0  HHHH  HHL  L
C  11  010  000  0  LLLL  HHL  L
C  11  110  000  0  LLLL  HHL  L
C  11  011  000  0  LLLL  HHL  H
C  11  001  000  0  LLLL  HHH  L
C  11  000  000  0  HHHL  HHH  L
C  11  100  101  0  HHHH  HHL  L
C  11  101  101  0  HHHH  HHL  H
C  00  111  101  0  HHHH  LHH  L
C  00  111  101  0  HHHH  LHH  L

```



Figure 6. Input File for Sequencer Section

```

/*****
/*
/*          CUPL INPUT FILE
/*          Design input for the sequencer section for the
/*          Dual Port DRAM Controller
/*
/*
/*****
/*          Allowable Target Device Types: GAL16V8
/*
/*****

PARTNO      SEQUENCER SECTION;
NAME        DRAM CONTROLLER;
DATE        03/28/86 ;
REV         01 ;
DESIGNER    Jerry Greiner;
COMPANY     Lattice Semiconductor;
ASSEMBLY    TWO;
LOCATION     U11;

/** Inputs **/

PIN 1       = SYSCLK;
PIN [2,3]   = [!RQGT A,!RQGT B] ;
PIN 4       = RDY ;
PIN 5       = !RFGT ;
PIN 6       = !WR ;
PIN 7       = ACK ;
PIN 8       = !RES ;
PIN 11      = !OE ;

/** Outputs **/

PIN 19      = ST2 ;
PIN 18      = ST1 ;
PIN 17      = ST0 ;
PIN 16      = DIR ;
PIN 15      = !CAD ;
PIN 14      = !RAD ;
PIN 13      = !ACKREF ;
PIN 12      = !WE ;

```

Figure 6. (cont'd)

```

/** Declarations and Intermediate Variable Definitions */

```

```

DST1 = ST2 & ST1 & ST0 ;
DST2 = !ST2 & ST1 & !ST0 ;
DST3 = ST2 & ST1 & !ST0 ;
DST4 = !ST2 & ST1 & ST0 ;
DST5 = !ST2 & !ST1 & ST0 ;
DST6 = !ST2 & !ST1 & !ST0 ;
DST7 = ST2 & !ST1 & !ST0 ;
DST8 = ST2 & !ST1 & ST0 ;

```

```

STCYC = ((RQGT A # RQGT B) & RDY # RFGT) ;

```

```

ST2.D = (DST2 # DST6 # DST8 # DST7 # DST4 & RFGT) # RES #
DST1 & !STCYC ;

```

```

ST1.D = DST2 # DST3 # DST8 # DST4 & RFGT # DST1 # RES ;

```

```

ST0.D = (DST3 # DST4 # DST7 # DST8) # RES # DST1 & !STCYC ;

```

```

DIR.D = WR & !DST1 ;

```

```

CAD.D = DST3 & !RFGT # DST4 & !RFGT # DST5 ;

```

```

RAD.D = (RQGT A # RQGT B) & RDY & (DST1 # DST2) ;

```

```

ACKREF = RFGT & ACK ;

```

```

WE.D = WR & (DST5 # DST6) ;

```

Figure 7. Simulation File for Sequencer Section

```

/*****
/*
/*          CUPL SIMULATION FILE          */
/*      Simulation File for the sequencer section of the      */
/*      Dual Port DRAM Controller                      */
/*
/*
/*          Allowable Target Device Types: GAL16V8          */
*****/

```

```

PARTNO    CONTROLLER SECTION;
NAME      DRAM CONTROLLER;
DATE      03/28/86 ;
REV       01 ;
DESIGNER   Jerry Greiner
COMPANY    Lattice Semiconductor ;
ASSEMBLY   TWO;
LOCATION     BEAVERTON, ORE;

```

## ORDER:

```

SYSCLK,%2,!RES,%2,!RQGT,!RQGTB,!RFGT,%2,RDY,!WR,ACK,%2,!OE,%4,
ST2,ST1,ST0,%2,DIR,%2,!CAD,!RAD,%2,!WE ;

```

## VECTORS:

```

$num" S      !!
$num" Y      RR!
$num" S      ! QQR
$num" C      R GGF R!A ! SSS D CR !
$num" L      E TTG DWC O TTT I AA W
$num" K      S ABT YRK E 210 R DD E
$num" -----

```

```

0 0 111 010 0 XXX X XX X
0 0 111 010 0 XXX X XX X
C 0 111 010 0 HHH L XH H
C 1 011 010 0 HHH L HH H
C 1 011 110 0 LHL L HL H
C 1 011 110 0 HHL L HL H
C 1 011 010 0 LHH L LH H
C 1 011 010 0 LLH L LH H
C 1 011 010 0 LLL L LH H
C 1 011 010 0 HLL L HH H
C 1 011 010 0 HLH L HH H
C 1 011 010 0 HHH L HH H
C 1 011 010 0 HHH L HH H
C 1 011 010 0 HHH L HH H

```

### DESCRIPTION

This application example is for the control and monitor of a three-story elevator. The elevator operates among the three stories in response to Call buttons on each of the floors, as well as inside the car itself. Directional priority is based on the last direction traveled, with a reversal possible on the second floor.

The simplified operation of an elevator and each of its three logic blocks is described in detail in Section 4 of this handbook. The reader is encouraged to examine the code and the explanations to fully understand the workings of the latch, control, and display logic.

A block diagram of the three-chip system is shown in Figure 1. The state-transition diagram of Figure 2 reveals the complexity of this apparently simple application.

Complete CUPL program listings are provided for each of the three GAL16V8 devices. For the latch-logic device, these include the design input file (Figure 3), the expanded product terms (Figure 4), the symbol table (Figure 5), the 'fuse' plot (Figure 6), the chip pinout diagram (Figure 7), the JEDEC fuse-map file (Figure 8), and the compile directives (Figure 9). The associated CUPL files for the control-logic device are shown in Figures 10-16, and those for the display-logic device are shown in Figures 17-23.

The compile directives are shown only for reference. Notice that the control logic file takes more than 9 minutes to complete — the complexity of the state machine, combined with a minimization level of two (which reduces the product-term count on each output to a minimum), makes this a tremendous algorithmic task. A human attempting this task could take hours or days to perform the full minimization. Moreover, multiple iterations for design revisions would be impossible by hand, due to the time constraints. With previous-generation assembler programs, the user had to supply reduced equations to the program — a next-to-impossible task for this project.

Be sure to examine the output file for the control-logic device. Use of the output polarity control, available in the GAL macrocell, allows both active-high and active-low equations side by side. □

5

Figure 1. Block Diagram

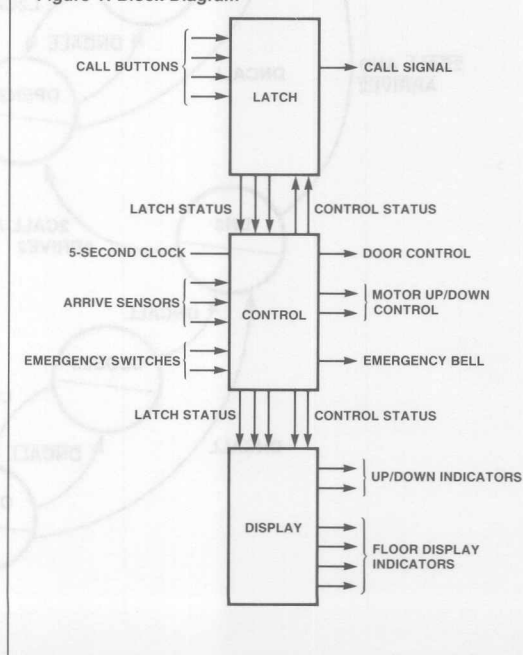


Figure 2. State Transition Diagram

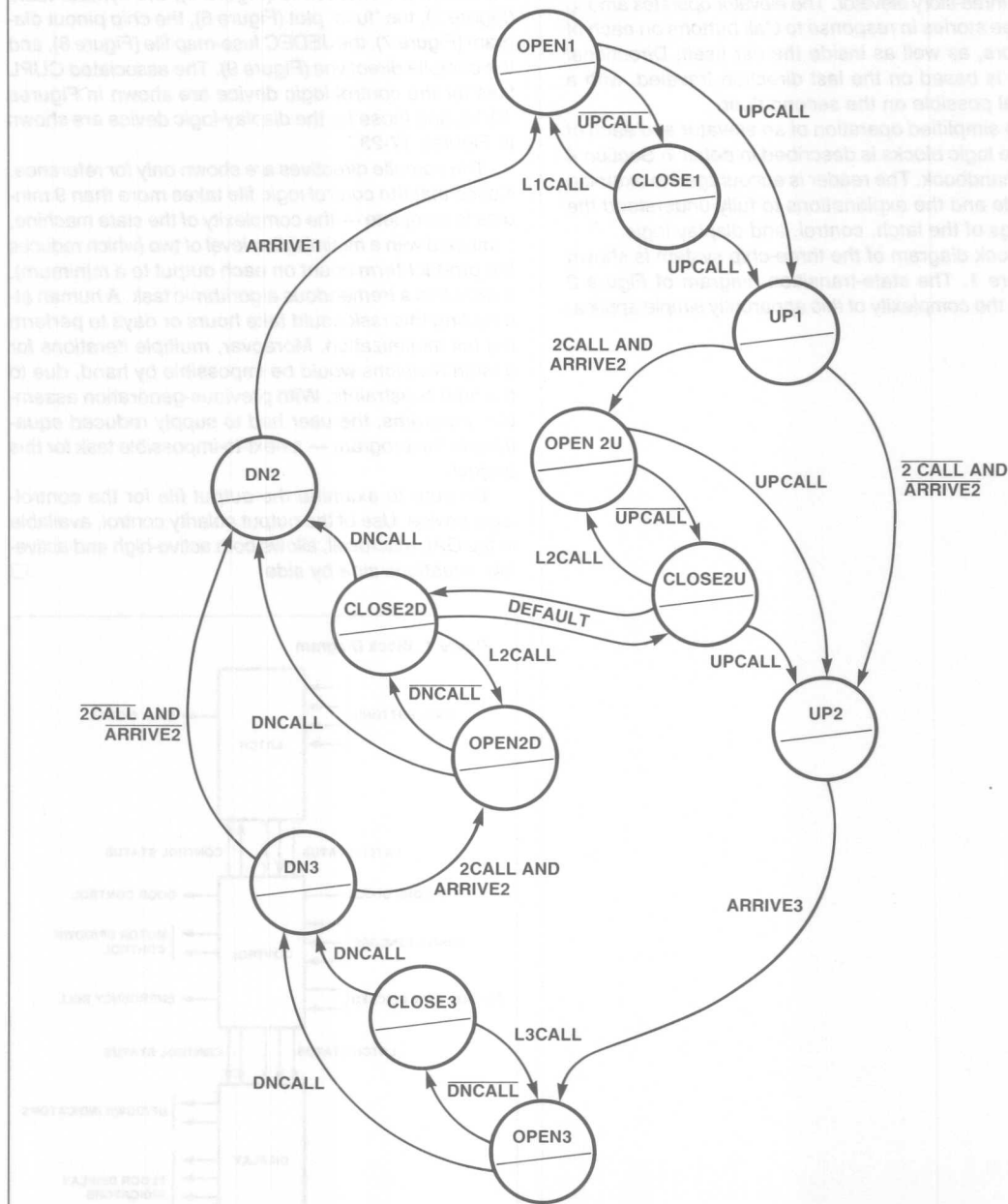


Figure 3. Design Input File for Latch Device

```

/*****
/*
/*      Three Story Elevator Example using a GAL16V8
/*
/*      Latch   Logic
/*
/*      CUPL Source File (Ele3_lat.PLD)
/*
*****/

PARTNO      01;
NAME        Ele3_lat;
REV         0;
DATE        4/20/86;
DESIGNER    Dean Suhr;
COMPANY     Lattice Semiconductor;
ASSEMBLY    Elevator Board;
LOCATION     U01;

/* device */

device g16v8;

/* inputs */

pin [1]      = [oCall];           /* Button in elevator */
pin [2,3,4]  = [1Call,2Call,3Call]; /* Buttons on Floors */
pin [5,6]    = [Door,Motion];    /* Control Status */
pin [7,8]    = [Fl_DirA,Fl_DirB]; /* 00 = 1Up    11 = 3Dn */
                                           /* 01 = 2Up    10 = 2Dn */

/* outputs */

pin [12]     = [Call];           /* Any Call active */
pin [13,14]  = [L1Call,L1Call_bar]; /* Latched call to 1st floor */
pin [15,16]  = [L2Call,L2Call_bar]; /* Latched call to 2nd floor */
pin [17,18]  = [L3Call,L3Call_bar]; /* Latched call to 3rd floor */

/* internal nodes */

Up = !Fl_DirA;           /* 00 = 1Up    11 = 3Dn */
Dn = Fl_DirA;           /* 01 = 2Up    10 = 2Dn */

1Floor = !Fl_DirA & !Fl_DirB;
2Floor = !1Floor & !3Floor;
3Floor = Fl_DirA & Fl_DirB;

/* logic equations */

Call = L1Call # L2Call # L3Call;

L1Call = !( L1Call_bar # 1Call # (1Floor & oCall));
L1Call_bar = !( L1Call # !Motion);

L2Call = !( L2Call_bar # 2Call # (2Floor & oCall));
L2Call_bar = !( L2Call # !Motion);

L3Call = !( L3Call_bar # 3Call # (3Floor & oCall));
L3Call_bar = !( L3Call # !Motion);

```



Figure 4. Expanded Product Terms for Latch Device

## Expanded Product Terms

```

=====
Expanded Product Terms
=====

1Floor =>
    !Fl_DirA & !Fl_DirB
2Floor =>
    !Fl_DirA & Fl_DirB
    # Fl_DirA & !Fl_DirB
3Floor =>
    Fl_DirA & Fl_DirB

Call =>
    L1Call
    # L2Call
    # L3Call

Dn =>
    Fl_DirA

L1Call =>
    L1Call_bar
    # 1Call
    # !Fl_DirA & !Fl_DirB & oCall
L1Call_bar =>
    L1Call
    # !Motion

L2Call =>
    L2Call_bar
    # 2Call
    # !Fl_DirA & Fl_DirB & oCall
    # Fl_DirA & !Fl_DirB & oCall

L2Call_bar =>
    L2Call
    # !Motion

L3Call =>
    L3Call_bar
    # 3Call
    # Fl_DirA & Fl_DirB & oCall
L3Call_bar =>
    L3Call
    # !Motion

Up =>
    !Fl_DirA

Call.oe =>
    1

L1Call.oe =>
    1

L1Call_bar.oe =>
    1

L2Call.oe =>
    1

L2Call_bar.oe =>
    1

L3Call.oe =>
    1

L3Call_bar.oe =>
    1

```

Figure 5. Symbol Table for Latch Device

Symbol Table

Pin	Variable				Pterms	Max	Min
Pol	Name	Ext	Pin	Type	Used	Pterms	Level
---	-----	---	---	---	---	---	---
	1Call		2	V	-	-	-
	1Floor		0	I	1	-	-
	2Call		3	V	-	-	-
	2Floor		0	I	2	-	-
	3Call		4	V	-	-	-
	3Floor		0	I	1	-	-
	Call		12	V	3	7	1
	Dn		0	I	1	-	-
	Door		5	V	-	-	-
	F1_DirA		7	V	-	-	-
	F1_DirB		8	V	-	-	-
	L1Call		13	V	3	7	1
	L1Call_bar		14	V	2	7	1
	L2Call		15	V	4	7	1
	L2Call_bar		16	V	2	7	1
	L3Call		17	V	3	7	1
	L3Call_bar		18	V	2	7	1
	Motion		6	V	-	-	-
	Up		0	I	1	-	-
	oCall		1	V	-	-	-
	Call	oe	12	D	1	1	0
	L1Call	oe	13	D	1	1	0
	L1Call_bar	oe	14	D	1	1	0
	L2Call	oe	15	D	1	1	0
	L2Call_bar	oe	16	D	1	1	0
	L3Call	oe	17	D	1	1	0
	L3Call_bar	oe	18	D	1	1	0

LEGEND    F : field            D : default variable            M : extended node  
           N : node            I : intermediate variable        T : function  
           V : variable        X : extended variable            U : undefined

Figure 6. 'Fuse' Plot for Latch Device

```

=====
                          Fuse Plot
=====

Syn   2192 - Ac0   2193 -

Pin #19  2048 Pol x  2120 Ac1 -
0000 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0032 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0064 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0096 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0128 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0160 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0192 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0224 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #18  2049 Pol x  2121 Ac1 -
0256 -----
0288 -----x-----
0320 -----x-----
0352 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0384 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0416 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0448 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0480 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #17  2050 Pol x  2122 Ac1 -
0512 -----
0544 -----x-----
0576 -----x-----
0608 --x-----x--x-----
0640 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0672 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0704 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0736 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #16  2051 Pol x  2123 Ac1 -
0768 -----
0800 -----x-----
0832 -----x-----
0864 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0896 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0928 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0960 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0992 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Pin #15  2052 Pol x  2124 Ac1 -
1024 -----
1056 -----x-----
1088 -----x-----
1120 --x-----x--x-----
1152 --x-----x--x-----
1184 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1216 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1248 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #14  2053 Pol x  2125 Ac1 -
1280 -----
1312 -----x-----
1344 -----x-----
1376 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1408 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1440 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1472 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1504 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #13  2054 Pol x  2126 Ac1 -
1536 -----
1568 -----x-----
1600 x-----
1632 --x-----x--x-----
1664 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1696 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1728 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1760 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #12  2055 Pol -  2127 Ac1 -
1792 -----
1824 -----x-----
1856 -----x-----
1888 -----x-----
1920 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1952 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1984 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
2016 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

```

LEGEND: X: Programmed Cell  
 -: Erased Cell

Figure 7. Pinout Diagram for Latch Device

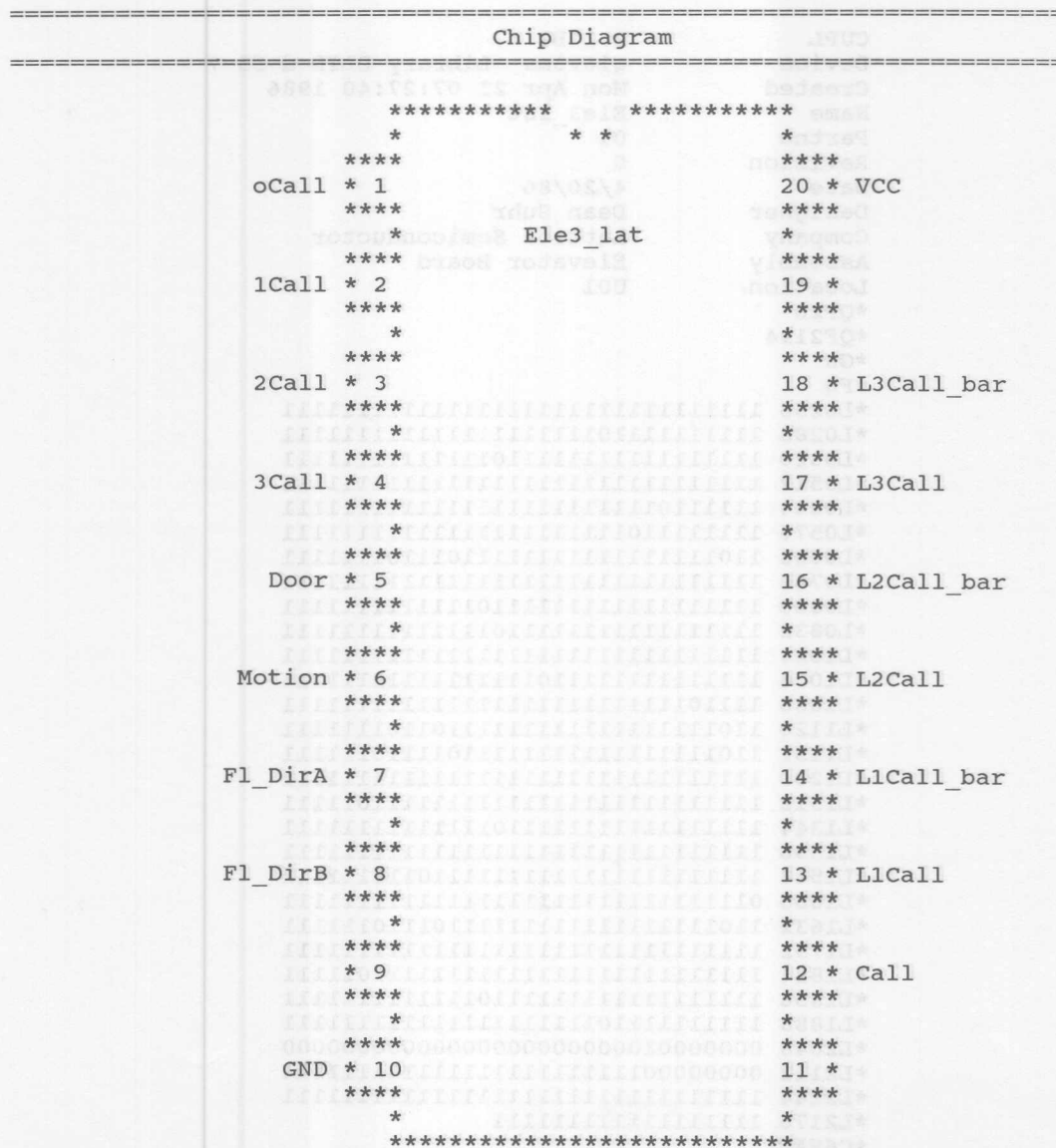


Figure 8. JEDEC File for Latch Device

```

CUPL                2.10B1
Device              gl6v8ma  Library DLIB-d-55-7
Created            Mon Apr 21 07:27:40 1986
Name               Ele3_lat
Partno             01
Revision           0
Date               4/20/86
Designer           Dean Suhr
Company            Lattice Semiconductor
Assembly           Elevator Board
Location           U01
*QP20
*QF2194
*G0
*F0
*L0256 11111111111111111111111111111111
*L0288 11111111110111111111111111111111
*L0320 1111111111111111111101111111111111
*L0512 1111111111111111111111111111111111
*L0544 1111110111111111111111111111111111
*L0576 1111111101111111111111111111111111
*L0608 1101111111111111111111011101111111
*L0768 1111111111111111111111111111111111
*L0800 1111111111111111111111011111111111
*L0832 1111111111111111111101111111111111
*L1024 1111111111111111111111111111111111
*L1056 1111111111111101111111111111111111
*L1088 1111011111111111111111111111111111
*L1120 1101111111111111111111101101111111
*L1152 1101111111111111111111011110111111
*L1280 1111111111111111111111111111111111
*L1312 111111111111111111111111111111011111
*L1344 1111111111111111110111111111111111
*L1536 1111111111111111111111111111111111
*L1568 1111111111111111111111011111111111
*L1600 0111111111111111111111111111111111
*L1632 1101111111111111111111011101111111
*L1792 1111111111111111111111111111111111
*L1824 1111111111111111111111111111011111
*L1856 1111111111111111111111011111111111
*L1888 1111111111011111111111111111111111
*L2048 0000000010000000000000000000000000
*L2112 0000000001111111111111111111111111
*L2144 1111111111111111111111111111111111
*L2176 111111111111111111
*C6FAC
*336A

```

Figure 9. Compile Directives for Latch Device

```
cupl -jlfx ele3_lat
```

```
CUPL Version 2.10B1
```

```
Copyright (c) 1983,84,85 Assisted Technology, Inc.
```

```
cuplx
```

```
time: 5 secs
```

```
cupla
```

```
time: 38 secs
```

```
cuplb
```

```
time: 10 secs
```

```
cuplm
```

```
time: 5 secs
```

```
cuplc
```

```
time: 16 secs
```

```
total time: 75 secs
```



Figure 10. Design Input File for Control Device

```

/*****
/*
/*      Three Story Elevator Example using a GAL16V8
/*
/*      Control Logic
/*
/*      CUPL Source File (Ele3_ctl.PLD)
*****/

PARTNO      02;
NAME        Ele3_ctl;
REV         0;
DATE        4/20/86;
DESIGNER    Dean Suhr;
COMPANY     Lattice Semiconductor;
ASSEMBLY    Elevator Board;
LOCATION     U02;

/* device */

device g16v8;

/* inputs */

pin [1,11]   = [CLK,!OE];
pin [2,3,4]  = [L1Call,L2Call,L3Call]; /* Latched call */
pin [5,6,7]  = [Arr1,Arr2,Arr3]; /* Floor arrival sensor */
pin [8,9]    = [Emerg_Stop,Emerg_Bell]; /* Emergency Switches */

/* outputs */

pin [12]      = [!Door]; /* 1 = Open, 0 = Close */
pin [13,14]   = [MotorUp,MotorDn]; /* Motor run signals */
pin [15,16]   = [Fl_DirA,Fl_DirB]; /* 00 = 1Up, 11 = 3Dn
/* 01 = 2Up, 10 = 2Dn */
pin [17,18]   = [MotionU,MotionD]; /* 0 = Wait, 1 = Move */
pin [19]      = [Bell]; /* 1 = Ring, 0 = Silent */

/* reduction level flags */

min door.d   = 2;
min Fl_DirA.d = 2;
min Fl_DirB.d = 2;

/* internal nodes */

Up = !Fl_DirA; /* 00 = 1Up 11 = 3Dn */
Dn = Fl_DirA; /* 01 = 2Up 10 = 2Dn */

1Floor = !Fl_DirA & !Fl_DirB;
2Floor = !1Floor & !3Floor;
3Floor = Fl_DirA & Fl_DirB;

```

Figure 10. (cont'd)

```

/* logic equations */

MotorUp = MotionU & !Emerg_Stop
        & !Fl_DirA & ( !Arr2 # !Arr3 # Arr2 & !L2Call);
MotorDn = MotionD & !Emerg_Stop
        & Fl_DirA & ( !Arr1 # !Arr2 # Arr2 & !L2Call);
Bell    = Emerg_Stop # Emerg_Bell;

/* State Definitions */

fld Control = [!Door, MotionU, MotionD, Fl_DirA, Fl_DirB];

$define    Open1    'b'10000
$define    Close1   'b'00000
$define    Up1      'b'01000
$define    Open2u   'b'10001
$define    Open2d   'b'10010
$define    Close2u  'b'00001
$define    Close2d  'b'00010
$define    Up2      'b'01001
$define    Dn2      'b'00110
$define    Open3    'b'10011
$define    Close3   'b'00011
$define    Dn3      'b'00111

sequence Control {

Present Open1:
    If ( L2Call # L3Call ) next Up1;
    default next Close1;

Present Close1:
    If ( L2Call # L3Call ) next Up1;
    If ( L1Call ) next Open1;
    default next Close1;

Present Up1:
    If (Arr2 & !L2Call ) next Up2;
    If (Arr2 & L2Call ) next Open2u;
    default next Up1;

Present Open2u:
    If ( L3Call ) next Up2;
    default next Close2u;

Present Close2u:
    If ( L3Call ) next Up2;
    If ( L2Call ) next Open2u;
    default next Close2d;

```

Figure 10. (cont'd)

```

Present Up2:
  If ( Arr3 ) next Open3;
  default next Up2;

Present Open2d:
  If ( L1Call ) next Dn2;
  default next Close2d;

Present Close2d:
  If ( L1Call ) next Dn2;
  If ( L2Call ) next Open2d;
  default next Close2u;

Present Dn2:
  If (Arr1) next Open1;
  default next Dn2;

Present Open3:
  If ( L1Call # L2Call ) next Dn3;
  default next Close3;

Present Close3:
  If ( L1Call # L2Call ) next Dn3;
  If ( L3Call ) next Open3;
  default next Close3;

Present Dn3:
  If (Arr2 & !L2Call ) next Dn2;
  If (Arr2 & L2Call ) next Open2d;
  default next Dn3;
}

```

Figure 11. Expanded Product Terms for Control Device

```

*****
Ele3_ctl
*****

CUPL                2.10B1
Device              gl16v8ms  Library DLIB-d-55-9
Created             Mon Apr 21 07:54:50 1986
Name                Ele3_ctl
Partno              02
Revision            0
Date                4/20/86
Designer            Dean Suhr
Company             Lattice Semiconductor
Assembly            Elevator Board
Location            U02

=====
Expanded Product Terms
=====

1Floor =>
    !Fl_DirA & !Fl_DirB

2Floor =>
    !Fl_DirA & Fl_DirB
    # Fl_DirA & !Fl_DirB

3Floor =>
    Fl_DirA & Fl_DirB

Bell =>
    Emerg_Stop
    # Emerg_Bell

Control =>
    !Door , MotionU , MotionD , Fl_DirA , Fl_DirB

Dn =>
    Fl_DirA

Door.d =>
    Arr2 & Door & Fl_DirA & Fl_DirB & L2Call & MotionD & !MotionU
    # Door & !Fl_DirA & !Fl_DirB & L1Call & !MotionD & !MotionU
    # Arr2 & Door & !Fl_DirA & !Fl_DirB & L2Call & !MotionD & MotionU
    # Door & !Fl_DirA & Fl_DirB & L2Call & !MotionD & !MotionU
    # Arr3 & Door & !Fl_DirA & Fl_DirB & !MotionD & MotionU
    # Door & Fl_DirA & !Fl_DirB & L2Call & !MotionD & !MotionU
    # Arr1 & Door & Fl_DirA & !Fl_DirB & MotionD & !MotionU
    # Door & Fl_DirA & Fl_DirB & L3Call & !MotionD & !MotionU

```

Figure 11. (cont'd)

```

Fl_DirA.d =>
    Door & Fl_DirA & Fl_DirB & !MotionU
    # !Door & Fl_DirA & !MotionD & !MotionU
    # Fl_DirA & L1Call & !MotionD & !MotionU
    # Fl_DirA & L2Call & !MotionD & !MotionU
    # !Arr1 & Door & Fl_DirA & MotionD & !MotionU
    # Arr3 & Door & !Fl_DirA & Fl_DirB & !MotionD & MotionU
    # Door & Fl_DirB & !L2Call & !L3Call & !MotionD & !MotionU

Fl_DirB.d =>
    !Door & Fl_DirB & !MotionD & !MotionU
    # Fl_DirA & Fl_DirB & !MotionD & !MotionU
    # Fl_DirB & L2Call & !MotionD & !MotionU
    # Fl_DirB & L3Call & !MotionD & !MotionU
    # Arr2 & Door & !Fl_DirA & !MotionD & MotionU
    # Door & !Fl_DirA & Fl_DirB & !MotionD & MotionU
    # !Arr2 & Door & Fl_DirA & Fl_DirB & !MotionU
    # Door & Fl_DirA & !L1Call & !L2Call & !MotionD & !MotionU

MotionD.d =>
    !Arr2 & Door & Fl_DirA & Fl_DirB & MotionD & !MotionU
    # Door & Fl_DirA & Fl_DirB & !L2Call & MotionD & !MotionU
    # !Arr1 & Door & Fl_DirA & !Fl_DirB & MotionD & !MotionU
    # Fl_DirA & L1Call & !MotionD & !MotionU
    # Fl_DirA & Fl_DirB & L2Call & !MotionD & !MotionU

MotionU.d =>
    !Fl_DirA & !Fl_DirB & L2Call & !MotionD & !MotionU
    # !Arr3 & Door & !Fl_DirA & Fl_DirB & !MotionD & MotionU
    # !Fl_DirA & L3Call & !MotionD & !MotionU
    # Door & !Fl_DirA & !Fl_DirB & !L2Call & !MotionD & MotionU
    # !Arr2 & Door & !Fl_DirA & !Fl_DirB & !MotionD & MotionU

MotorDn =>
    !Arr1 & !Emerg_Stop & Fl_DirA & MotionD
    # !Arr2 & !Emerg_Stop & Fl_DirA & MotionD
    # Arr2 & !Emerg_Stop & Fl_DirA & !L2Call & MotionD

MotorUp =>
    !Arr2 & !Emerg_Stop & !Fl_DirA & MotionU
    # !Arr3 & !Emerg_Stop & !Fl_DirA & MotionU
    # Arr2 & !Emerg_Stop & !Fl_DirA & !L2Call & MotionU

Up =>
    !Fl_DirA

Bell.oe =>
    1

MotorDn.oe =>
    1

MotorUp.oe =>
    1

```

Figure 12. Symbol Table for Control Device

Symbol Table							
Pin Pol	Variable Name	Ext	Pin	Type	Pterms Used	Max Pterms	Min Level
	1Floor		0	I	1	-	-
	2Floor		0	I	2	-	-
	3Floor		0	I	1	-	-
	Arr1		5	V	-	-	-
	Arr2		6	V	-	-	-
	Arr3		7	V	-	-	-
	Bell		19	V	2	7	1
	CLK		1	V	-	-	-
	Control		0	F	-	-	-
	Dn		0	I	1	-	-
!	Door		12	V	-	-	-
!	Door	d	12	X	8	8	1
	Emerg_Bell		9	V	-	-	-
	Emerg_Stop		8	V	-	-	-
	F1_DirA		15	V	-	-	-
	F1_DirA	d	15	X	7	8	2
	F1_DirB		16	V	-	-	-
	F1_DirB	d	16	X	8	8	2
	L1Call		2	V	-	-	-
	L2Call		3	V	-	-	-
	L3Call		4	V	-	-	-
	MotionD		18	V	-	-	-
	MotionD	d	18	X	5	8	1
	MotionU		17	V	-	-	-
	MotionU	d	17	X	5	8	1
	MotorDn		14	V	3	7	1
	MotorUp		13	V	3	7	1
!	OE		11	V	-	-	-
	Up		0	I	1	-	-
	door	d	0	I	-	-	-
	Bell	oe	19	D	1	1	0
	MotorDn	oe	14	D	1	1	0
	MotorUp	oe	13	D	1	1	0

LEGEND    F : field            D : default variable            M : extended node  
              N : node            I : intermediate variable        T : function  
              V : variable        X : extended variable            U : undefined



Figure 13. 'Fuse' Plot for Control Device

```

=====
                        Fuse Plot
=====
Syn      2192 x Ac0      2193 -

Pin #19  2048  Pol - 2120  Ac1 -
0000 -----
0032 -----x-----
0064 -----x-----
0096 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0128 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0160 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0192 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0224 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #18  2049  Pol - 2121  Ac1 x
0256 -----x---x---x---xx-----x
0288 -----xx---x---x---x-----x
0320 -----x---x---x---x-----x
0352 x-----x---x-----x-----
0384 -----x---x---x---x-----
0416 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0448 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0480 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #17  2050  Pol - 2122  Ac1 x
0512 -----x---x---x---x-----
0544 -----x---x---x---x-----x
0576 -----xx---x-----x-----
0608 -----x---x---x---x-----x
0640 -----x---x---x---x-----x
0672 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0704 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0736 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #16  2051  Pol - 2123  Ac1 x
0768 -----x---x---x-----x-
0800 -----x---x---x-----
0832 -----x---x---x-----
0864 -----xx---x---x-----
0896 -----x---x---x-----x
0928 -----x---x---x-----x
0960 -----x---x---xx-----x
0992 -x---x---x-----x-

Pin #15  2052  Pol - 2124  Ac1 x
1024 -----x---x---x-----x
1056 -----x---x---x-----x-
1088 x-----x---x-----x-----
1120 ---x---x---x-----x-----
1152 -----x---x---x-----x
1184 -----x---x---xx-----x
1216 -----x---x---x---x-----x
1248 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #14  2053  Pol - 2125  Ac1 -
1280 -----
1312 -----x-----x-----x-----
1344 -----x-----xx-----x-----
1376 -----xx-----x---x-----x-----
1408 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1440 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1472 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1504 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #13  2054  Pol - 2126  Ac1 -
1536 -----
1568 -----x-----x---x-----x-----
1600 -----x-----x---x-----x-----
1632 -----x-----x---x-----x-----
1664 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1696 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1728 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1760 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #12  2055  Pol - 2127  Ac1 x
1792 ---x---x---x---x---x-----x
1824 x-----x---x---x-----x-----
1856 ---x---x---x---xx---x-----x
1888 ---x---x---x---x---x-----x
1920 -----x---x---x---xx-----x
1952 ---x---x---x---x---x-----x
1984 ---x---xx---x---x-----x
2016 -----xx---x---x---x-----x

```

LEGEND:    X: Programmed Cell  
          -: Erased Cell

Figure 14. Pinout Diagram for Control Device

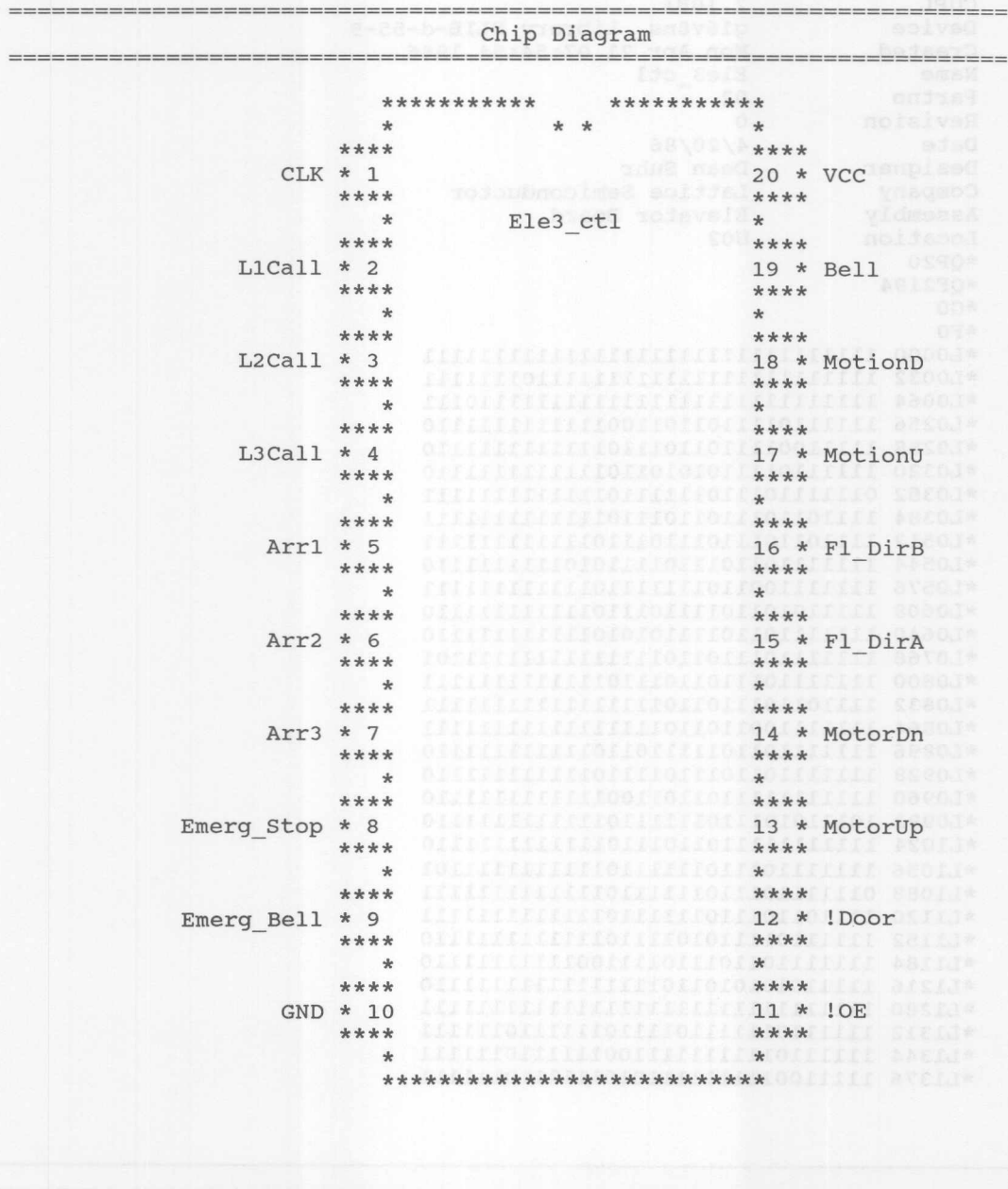




Figure 15. (cont'd)

```

*L1536 1111111111111111111111111111111111
*L1568 1111111111011111101011111011111111
*L1600 1111111110111111101011101111111111
*L1632 1111101111011111010111110111111111
*L1792 111101011110101010111111111111110
*L1824 011111101110111011111111111111110
*L1856 111101010101111001011111111111110
*L1888 111101011101010111101111111111110
*L1920 111111101010111011110011111111110
*L1952 111101011101101101101111111111110
*L1984 111110111110010101011111111111110
*L2016 111111001010101101111111111111110
*L2048 1111111000000000000000000000000000
*L2112 0000000010000110111111111111111111
*L2144 1111111111111111111111111111111111
*L2176 111111111111111101
*C9228
*BB8C

```

Figure 16. Compile Directives for Control Device

```

cupl -jlfx ele3_ctl

CUPL Version 2.10B1
Copyright (c) 1983,84,85 Assisted Technology, Inc.

cuplx
time: 8 secs
cupla
time: 230 secs
cuplb
time: 52 secs
cuplm
time: 237 secs
cuplc
time: 21 secs
total time: 550 secs

```

Figure 17. Design Input File for Display Device

```

/*****
/*
/*      Three Story Elevator Example using a GAL16V8
/*
/*      Display Logic
/*
/*      CUPL Source File (Ele3_dsp.PLD)
/*
*****/
PARTNO      03;
NAME        Ele3_dsp;
REV         0;
DATE        4/20/86;
DESIGNER    Dean Suhr;
COMPANY     Lattice Semiconductor;
ASSEMBLY    Elevator Board;
LOCATION     U03;

/* device */
device gl6v8;

/* inputs */

pin [1,2,3]   = [L1Call,L2Call,L3Call]; /* Call Status
pin [4,5]     = [Fl_DirA,Fl_DirB];      /* 00 = 1Up, 11 = 3Dn
/* 01 = 2Up, 10 = 2Dn

/* outputs */
/*      DISPLAY DIAGRAMS
pin [12]      = [UpArrow];              /*      ^
pin [14]      = [DnArrow];              /*      v

pin [15,16]   = [Seg13,Seg123];          /*      23
pin [17,18]   = [Seg2,Seg23];            /*      ----
/*      | 123
/*      23 |
/*      ---
/*      2 | | 13
/*      | |
/*      ---
/*      23

/* internal nodes */

Up = !Fl_DirA; /* 00 = 1Up 11 = 3Dn
Dn = Fl_DirA;  /* 01 = 2Up 10 = 2Dn

1Floor = !Fl_DirA & !Fl_DirB;
2Floor = !1Floor & !3Floor;
3Floor = Fl_DirA & Fl_DirB;

```

Figure 17. (cont'd)

```

/* logic equations */

UpArrow = Up & ( L2Call # L3Call);
DnArrow = Dn & ( L1Call # L2Call);

/*          TABLE INPUTS          TABLE OUTPUTS          */
table 1Floor,2Floor,3Floor => Seg13, Seg123, Seg2, Seg23 {
    'b'100          => 'b'1100;
    'b'010          => 'b'0111;
    'b'001          => 'b'1101; }

```



Figure 18. Expanded Product Terms for Display Device

```
*****
                                Ele3_dsp
*****
```

```
CUPL          2.10B1
Device        g16v8s  Library DLIB-d-55-8
Created       Mon Apr 21 08:02:40 1986
Name          Ele3_dsp
Partno        03
Revision      0
Date          4/20/86
Designer      Dean Suhr
Company       Lattice Semiconductor
Assembly      Elevator Board
Location      U03
```

```
=====
Expanded Product Terms
=====
```

```
1Floor =>
    !Fl_DirA & !Fl_DirB

2Floor =>
    !Fl_DirA & Fl_DirB
    # Fl_DirA & !Fl_DirB

3Floor =>
    Fl_DirA & Fl_DirB

Dn =>
    Fl_DirA

DnArrow =>
    Fl_DirA & L1Call
    # Fl_DirA & L2Call

Seg123 =>
    1

Seg13 =>
    0

Seg2 =>
    1

Seg23 =>
    0

Up =>
    !Fl_DirA

UpArrow =>
    !Fl_DirA & L2Call
    # !Fl_DirA & L3Call
```

Figure 19. Symbol Table for Display Device

Symbol Table							
Pin	Variable						
Pol	Name	Ext	Pin	Type	Pterms Used	Max Pterms	Min Level
---	-----	---	---	----	-----	-----	-----
	1Floor		0	I	1	-	-
	2Floor		0	I	2	-	-
	3Floor		0	I	1	-	-
	Dn		0	I	1	-	-
	DnArrow		14	V	2	8	1
	F1_DirA		4	V	-	-	-
	F1_DirB		5	V	-	-	-
	L1Call		1	V	-	-	-
	L2Call		2	V	-	-	-
	L3Call		3	V	-	-	-
	Seg123		16	V	1	8	1
	Seg13		15	V	1	8	1
	Seg2		17	V	1	8	1
	Seg23		18	V	1	8	1
	Up		0	I	1	-	-
	UpArrow		12	V	2	8	1
LEGEND							
F : field		D : default variable		M : extended node			
N : node		I : intermediate variable		T : function			
V : variable		X : extended variable		U : undefined			

Figure 20. 'Fuse' Plot for Display Device

```

=====
                          Fuse Plot
=====

Syn    2192 - Ac0    2193 x

Pin #19  2048  Pol x  2120  Ac1 -
0000 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0032 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0064 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0096 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0128 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0160 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0192 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0224 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #18  2049  Pol -  2121  Ac1 x
0256 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0288 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0320 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0352 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0384 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0416 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0448 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0480 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #17  2050  Pol -  2122  Ac1 x
0512 -----
0544 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0576 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0608 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0640 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0672 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0704 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0736 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #16  2051  Pol -  2123  Ac1 x
0768 -----
0800 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0832 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0864 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0896 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0928 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0960 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0992 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Pin #15  2052  Pol -  2124  Ac1 x
1024 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1056 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1088 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1120 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1152 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1184 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1216 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1248 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #14  2053  Pol -  2125  Ac1 x
1280 --x-----x-----
1312 x-----x-----
1344 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1376 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1408 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1440 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1472 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1504 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #13  2054  Pol x  2126  Ac1 -
1536 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1568 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1600 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1632 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1664 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1696 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1728 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1760 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #12  2055  Pol -  2127  Ac1 x
1792 x-----x-----
1824 ----x----x-----
1856 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1888 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1920 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1952 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1984 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
2016 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

```

LEGEND:    X: Programmed Cell  
           -: Erased Cell

Figure 21. Pinout Diagram for Display Device

## Chip Diagram

```

*****
*
L1Call * 1      20 * VCC
*
Ele3_dsp
*
L2Call * 2      19 *
*
L3Call * 3      18 * Seg23
*
Fl_DirA * 4     17 * Seg2
*
Fl_DirB * 5     16 * Seg123
*
* 6           15 * Seg13
*
* 7           14 * DnArrow
*
* 8           13 *
*
* 9           12 * UpArrow
*
GND * 10       11 *
*
*****

```

Figure 22. JEDEC File for Display Device

```

CUPL          2.10B1
Device        gl6v8s  Library DLIB-d-55-8
Created       Mon Apr 21 08:02:42 1986
Name          Ele3_dsp
Partno        03
Revision      0
Date          4/20/86
Designer      Dean Suhr
Company       Lattice Semiconductor
Assembly      Elevator Board
Location      U03
*QP20
*QF2194
*G0
*F0
*L0512 11111111111111111111111111111111
*L0768 11111111111111111111111111111111
*L1280 11011111011111111111111111111111
*L1312 01111111011111111111111111111111
*L1792 01111111011111111111111111111111
*L1824 11110111101111111111111111111111
*L2048 01111101000000000000000000000000
*L2112 00000000100000101111111111111111
*L2144 11111111111111111111111111111111
*L2176 11111111111111110
*C20C4
*9AF2

```

Figure 23. Compile Directives for Display Device

```
cupl -jlfx ele3_dsp

CUPL Version 2.10B1
Copyright (c) 1983,84,85 Assisted Technology, Inc.

cuplx
time: 5 secs
cupla
time: 41 secs
cuplb
time: 21 secs
cuplm
time: 5 secs
cuplc
time: 16 secs
total time: 89 secs
```





<b>INTRODUCTION</b>	<b>1</b>
<b>GAL DEVICE SPECIFICATIONS</b>	<b>2</b>
<b>LOGIC TUTORIAL</b>	<b>3</b>
<b>USING DEVELOPMENT TOOLS</b>	<b>4</b>
<b>GAL DEVICE APPLICATIONS</b>	<b>5</b>
<b>TECHNICAL BRIEFS</b>	<b>6</b>
<b>E<sup>2</sup>CMOS TECHNOLOGY OVERVIEW</b>	<b>7</b>
<b>GAL DEVICE QUALITY AND RELIABILITY</b>	<b>8</b>
<b>ARTICLE REPRINTS</b>	<b>9</b>
<b>APPENDICES</b>	<b>10</b>
<b>SALES OFFICES</b>	<b>11</b>



## INTRODUCTION

The generic architecture of the GAL family offers the user many different device configurations. One particular subset of these myriad architectural possibilities is common PAL architectures. GAL devices are capable of emulating all common PAL architectures with 100% pin, fuse-map, function, and parametric compatibility. In short, a GAL device can drop right into any PAL socket. This technical brief addresses the procedure of copying a PAL pattern — either a PAL master device or a PAL JEDEC file — into a GAL device. The technique is straightforward, since existing files or masters can be used without modification.

The first step in copying a PAL pattern into a GAL device is to determine whether the GAL device can emulate the particular architecture in question. This is accomplished by referencing Tables 1 and 2. Table 1 lists all of the 20-pin PAL architectures available as a subset of 20-pin GAL16V8 configurations. Likewise, Table 2 lists all of the 24-pin PAL architectures available as a subset of the 24-pin GAL20V8 configurations. The user must simply cross-reference the PAL architecture in question with these tables to determine which GAL device is needed.

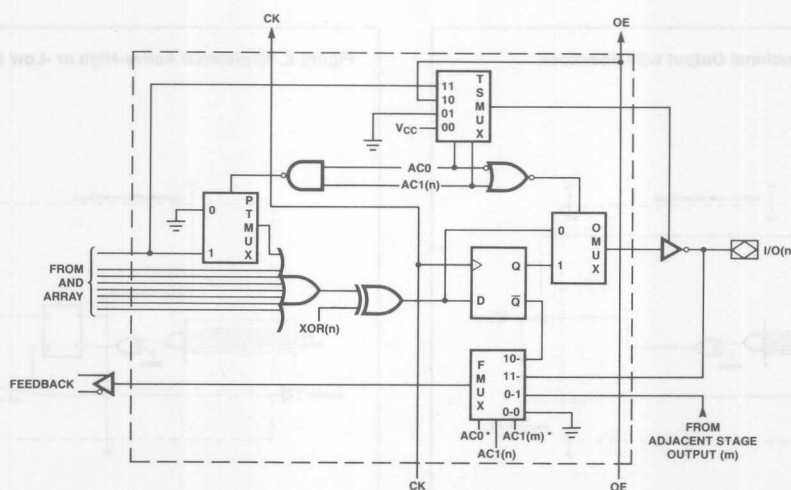
Table 1. PAL Architectures Emulated by GAL16V8

GAL16V8							
10L8	12L6	14L4	16L2	16R8	16R6	16R4	16L8
10H8	12H6	14H4	16H2	16RP8	16RP6	16RP4	16H8
10P8	12P6	14P4	16P2				16P8

Table 2. PAL Architectures Emulated by GAL20V8

GAL20V8							
14L8	16L6	18L4	20L2	20R8	20R6	20R4	20L8
14H8	16H6	18H4	20H2	20RP8	20RP6	20RP4	20H8
14P8	16P6	18P4	20P2				20P8

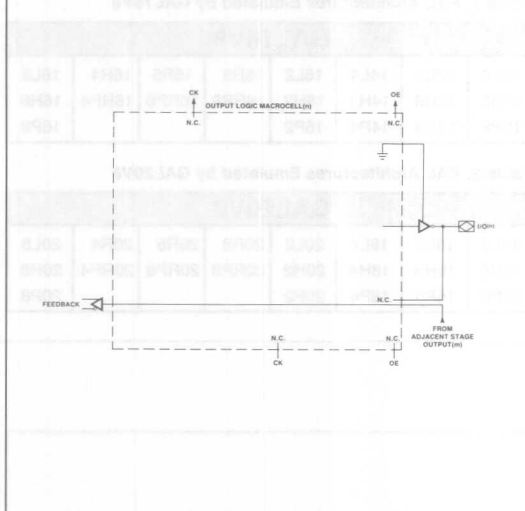
Figure 1. Output Logic Macrocell (OLMC)



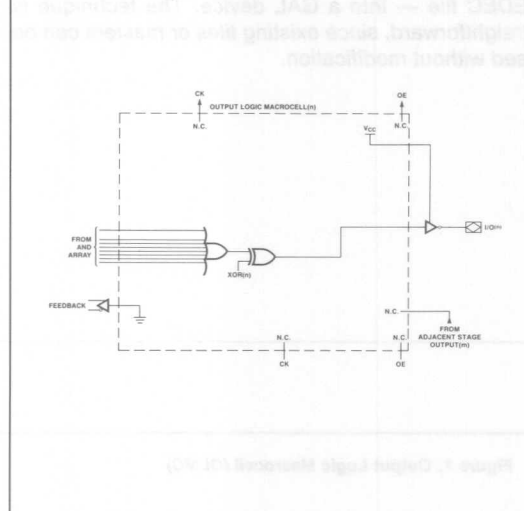
Function compatibility is an obvious requirement for copying to be performed. GAL devices are designed with a versatile Output Logic Macrocell (OLMC) that allows emulation of more than twenty different PAL architectures. The OLMC, shown in Figure 1, can be programmed to any of the configurations shown in Figures 2 through

5, namely: dedicated input, dedicated combinational output with programmable polarity, combinational output with feedback and programmable polarity, or registered output with feedback and programmable polarity. These four macrocell configurations can be combined as needed to allow full pin and function compatibility with the PAL architectures listed in the preceding tables.

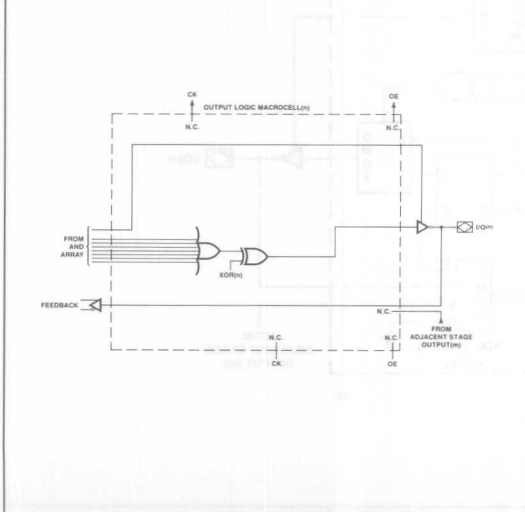
**Figure 2. Dedicated Input**



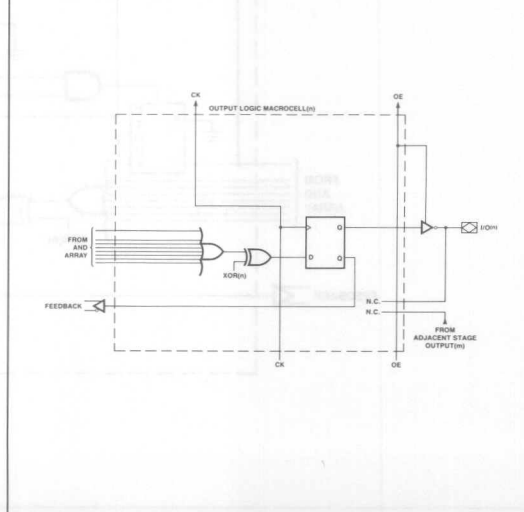
**Figure 3. Dedicated Combinational Output**



**Figure 4. Combinational Output with Feedback**



**Figure 5. Registered Active-High or -Low Output**



## Fuse-Map Compatibility

The ability of GAL devices to directly accept PAL patterns is the result of more than just functional compatibility. There is an additional stipulation that the 'fuse' maps be compatible — otherwise the copy procedure requires manual intervention in the form of manipulating JEDEC files or 'fuse' plots to get everything into the proper format. Obviously, any manual intervention would be cumbersome and time-consuming, and would significantly detract from the utility of the GAL devices.

Therefore, the GAL cell array is designed to be an identical bit-for-bit mapping of a PAL fuse array, in any output configuration. This means that external input signals and feedback signals are physically connected to the same array input lines in the GAL device as they are in a PAL device. Although this may sound trivial,

consider the following example, which examines how different PAL architectures have different array hook-up schemes.

Refer to the partial logic diagrams of a PAL16L2 and a PAL16R8 in Figures 6 and 7, respectively. Notice in the first figure that pin 1 is an input and is connected to array input lines 2 and 3. Now notice in the second figure that pin 1 is a clock input, and the feedback signal from pin 19 is connected to input lines 2 and 3. These differences pose no problem for the PAL manufacturer, since different PAL devices are manufactured independently of each other. However, for a GAL device to have the flexibility to accommodate different array hook-up schemes required some innovative design techniques. Although these techniques add to the complexity of the OLMC, the benefits of a directly copyable device are obvious.

Figure 6. Partial Logic Diagram of a PAL16L2 Device

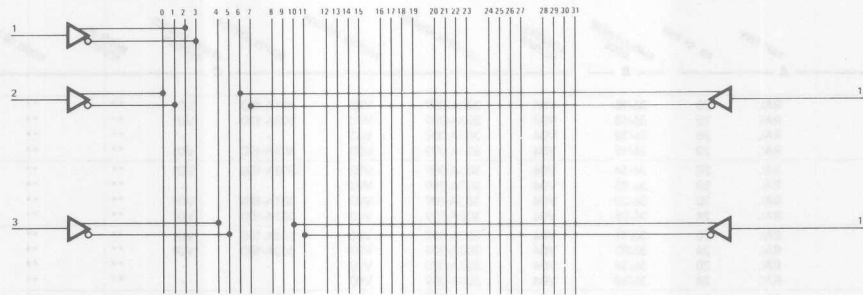
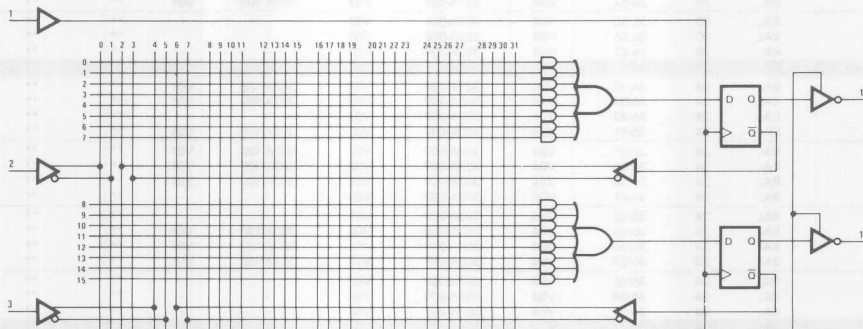


Figure 7. Partial Logic Diagram of PAL16R8 Device





# RAL Device Codes

The final element required to facilitate direct PAL device copying is the RAL (Reprogrammable Array Logic) code. The RAL code is the user's means of informing the programmer box exactly which architecture the GAL device will be patterned to. It correlates to the family/pin-code selection for identifying PAL architectures.

The information shown in Figure 8 is taken from the Data I/O wall chart listing all device manufacturers' family/pin codes. (Other hardware manufacturers offer similar support.) Notice that there is an entry for each of the GAL16V8 and the GAL20V8, as well as for many RAL codes. It should be noted that, although there are only two physically different device types — the GAL16V8 and GAL20V8 — these products source all of the 44

device types listed in Figure 10; the RAL codes are provided merely for the purpose of copying PAL devices. To emulate a PAL16R4 pattern using the GAL16V8, the RAL16R4 family/pin code must be used according to the procedure explained below. The RAL codes directly correspond to each respective PAL device configuration and provide the programmer box with the information to program the GAL device OLMCs.

Physically resident inside the GAL device is an 82-bit architecture control word which, once programmed, determines the configuration of the OLMCs. (Refer to the data sheets in Section 2 for further discussion on the architecture control word.) Downloading a PAL pattern to a programmer box transfers array data only; no information as to the device architecture is transmitted. Thus, the RAL code is essential.

Figure 8. GAL Device Pinout Codes

PROGRAMMABLE LOGIC											
DEVICE	PART TYPE	NO. OF PINS	FAMILY/PINOUT	LOGICPAK REVISION	LOGICPAK ADAPTER	ADAPTER REVISION	DESIGN ADAPTER	DESIGN ADAPTER REVISION	MODEL 50 REVISION	MODEL 60 ADAPTER	ABEL VERSION
	A		B				C				
10H8-25/-35	RAL	20	36/18	V04	303A-009	V03	303A-100	V01	**	**	1.0
10L8-25/-35	RAL	20	36/13	V04	303A-009	V03	303A-100	V01	**	**	1.0
10P8-25/-35	RAL	20	36/32	V04	303A-009	V03	-	-	**	**	1.0
12H6-25/-35	RAL	20	36/19	V04	303A-009	V03	303A-100	V01	**	**	1.0
12L6-25/-35	RAL	20	36/14	V04	303A-009	V03	303A-100	V01	**	**	1.0
12P6-25/-35	RAL	20	36/33	V04	303A-009	V03	-	-	**	**	1.0
14H4-25/-35	RAL	20	36/20	V04	303A-109	V03	303A-100	V01	**	**	1.0
14H8-25/-35	RAL	24	36/08	V04	303A-009	V03	303A-100	V01	**	**	1.0
14L4-25/-35	RAL	20	36/15	V04	303A-009	V03	303A-100	V01	**	**	1.0
14L8-25/-35	RAL	24	36/02	V04	303A-009	V03	303A-100	V01	**	**	1.0
14P4-25/-35	RAL	20	36/34	V04	303A-009	V03	-	-	**	**	1.0
14P8-25/-35	RAL	24	36/38	V04	303A-009	V03	-	-	**	**	1.0
16H2-25/-35	RAL	20	36/22	V04	303A-009	V03	303A-100	V01	**	**	1.0
16H6-25/-35	RAL	24	36/09	V04	303A-009	V03	303A-100	V01	**	**	1.0
16H8-25/-35	RAL	20	36/25	V04	303A-009	V03	303A-100	V01	**	**	1.0
16L2-25/-35	RAL	20	36/16	V04	303A-009	V03	303A-100	V01	**	**	1.0
16L6-25/-35	RAL	24	36/03	V04	303A-009	V03	303A-100	V01	**	**	1.0
16L8-25/-35	RAL	20	36/17	V04	303A-009	V03	303A-100	V01	**	**	1.0
16P2-25/-35	RAL	20	36/35	V04	303A-009	V03	-	-	**	**	1.0
16P6-25/-35	RAL	24	36/39	V04	303A-009	V03	-	-	**	**	1.0
16P8-25/-35	RAL	20	36/30	V04	303A-009	V03	-	-	**	**	1.0
16R4-25/-35	RAL	20	36/81	V04	303A-009	V03	303A-100	V01	**	**	1.0
16R6-25/-35	RAL	20	36/80	V04	303A-009	V03	303A-100	V01	**	**	1.0
16R8-25/-35	RAL	20	36/82	V04	303A-009	V03	303A-100	V01	**	**	1.0
16RP4-25/-35	RAL	20	36/85	V04	303A-009	V03	-	-	**	**	1.0
16RP6-25/-35	RAL	20	36/86	V04	303A-009	V03	-	-	**	**	1.0
16RP8-25/-35	RAL	20	36/87	V04	303A-009	V03	-	-	**	**	1.0
16V8-25/-35	GAL	20	36/55	V04	303A-009	V03	-	-	**	**	1.13
18H4-25/-35	RAL	24	36/10	V04	303A-009	V03	303A-100	V01	**	**	1.0
18L4-25/-35	RAL	24	36/04	V04	303A-009	V03	303A-100	V01	**	**	1.0
18P4-25/-35	RAL	24	36/40	V04	303A-009	V03	-	-	**	**	1.0
20H2-25/-35	RAL	24	36/11	V04	303A-009	V03	303A-100	V01	**	**	1.0
20H8-25/-35	RAL	24	36/61	V04	303A-009	V03	303A-100	V01	**	**	1.0
20L2-25/-35	RAL	24	36/05	V04	303A-009	V03	303A-100	V01	**	**	1.0
20L8-25/-35	RAL	24	36/26	V04	303A-009	V03	303A-100	V01	**	**	1.0
20P2-25/-35	RAL	24	36/41	V04	303A-009	V03	-	-	**	**	1.0
20P8-25/-35	RAL	24	36/62	V04	303A-009	V03	-	-	**	**	2.0
20R4-25/-35	RAL	24	36/65	V04	303A-009	V03	303A-100	V01	**	**	1.0
20R6-25/-35	RAL	24	36/66	V04	303A-009	V03	303A-100	V01	**	**	1.0
20R8-25/-35	RAL	24	36/27	V04	303A-009	V03	303A-100	V01	**	**	1.0
20RP4-25/-35	RAL	24	36/46	V04	303A-009	V03	-	-	**	**	2.0
20RP6-25/-35	RAL	24	36/64	V04	303A-009	V03	-	-	**	**	2.0
20RP8-25/-35	RAL	24	36/63	V04	303A-009	V03	-	-	**	**	2.0
20V8-25/-35	GAL	24	36/57	V04	303A-009	V03	-	-	**	**	2.0

The “...” symbol means that the algorithm for this device is under evaluation for possible addition in some future update.

Inside the memory of a device programmer is the predetermined GAL architecture control word corresponding to each PAL device configuration. By selecting the appropriate RAL code, the device programmer can append the predetermined architecture control word to the array information previously downloaded. It stands to reason that if the GAL family/pin code were selected, the GAL device's array would be programmed properly but the OLMCs would be left unprogrammed, since there is no predetermined OLMC configuration in a 'virgin' GAL device.

The GAL family/pin code would only be used in the

development of a new design from scratch. In that case, the JEDEC file generated by the development software would contain an architecture control word specific to the design in question. When copying an existing PAL master device or an existing PAL JEDEC file, the architecture control word only gets physically appended to the cell array information by the programmer box when the appropriate RAL code is selected.

A final reminder: When copying PAL device patterns into GAL devices, use the RAL family/pin code. When beginning a GAL design from scratch, use the GAL family/pin code.

## ACTUAL STEPS FOR COPYING PAL DEVICE PATTERNS INTO GAL DEVICES

This procedure is generic in nature and is not specific to any device programmer. A PAL master device or a PAL JEDEC file may be used.

- 1 Load either the PAL-device fuse map or JEDEC-file data into the device programmer memory using the normal procedure.
- 2 Select the appropriate RAL family/pin code from the programmer chart. Note: Remember that the RAL code is required for PAL device copying in order to configure the OLMCs properly. Do not use the GAL device code for copying PAL devices.

(Note: Some programmers require changing adapters when switching from one device manufacturer to another.)

- 3 Program the GAL device using the appropriate RAL family/pin code to configure to OLMCs properly. The copy procedure is complete and the resulting GAL device is 100% compatible with the PAL device it copied.

(Note: The GAL device still retains its full erasability feature. The device can be reused with different array patterns and architecture configurations (RAL codes), as selected by the designer.)

## INTRODUCTION

At Lattice Semiconductor, GAL devices are 'Anything, Everytime, Instantly.' 'Instantly' signifies very fast (340ms) programming time and even faster (50ms) erase time, allowing instant reprogramming and reconfiguring. 'Everytime' signifies GAL devices' unmatched quality, backed by guaranteed 100% programming and functional yields. 'Anything' signifies generic architecture and the ability to put any function on any pin.

'Anything' implies not only the many PAL architectures that GAL devices emulate with 100% socket compatibility, and not only in-between architectures (such as a 16R1 or 13L5), but also the ability to begin with an industry-standard architecture, say a 16R4, and have the flexibility of moving any function to any pin to simplify board layout. This brief explores the generic architecture and how it can be applied to designs already fixed in architecture that require greater flexibility for board layout.

The GAL16V8 and GAL20V8 are each capable of emulating 21 different PAL architectures, as shown in Tables 1 and 2. A GAL device can drop right into any of these existing PAL sockets with 100% compatibility including: pin-for-pin, function, parametric, and fuse-map compatibility. The reader is encouraged to reference the preceding brief and other sections of this handbook for a more comprehensive discussion of compatibility.

### In-Between Architectures

In addition to the industry standard architectures, the generic approach allows GAL devices to be configured to all the 'in-between' architectures not offered by PAL device manufacturers. For instance, a 16R1 or 20R7 —

Table 1. PAL Architectures Emulated by GAL16V8

GAL16V8							
10L8	12L6	14L4	16L2	16R8	16R6	16R4	16L8
10H8	12H6	14H4	16H2	16RP8	16RP6	16RP4	16H8
10P8	12P6	14P4	16P2				16P8

Table 2. PAL Architectures Emulated by GAL20V8

GAL20V8							
14L8	16L6	18L4	20L2	20R8	20R6	20R4	20L8
14H8	16H6	18H4	20H2	20RP8	20RP6	20RP4	20H8
14P8	16P6	18P4	20P2				20P8

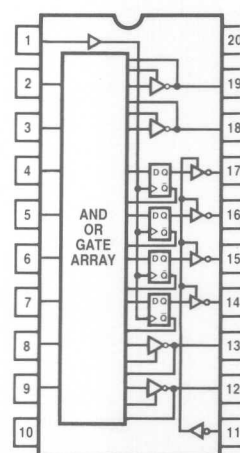
or even a 13L5 — are possible by simply specifying the function of each pin in the software design input. The obvious benefit lies in not being confined to a specific architecture during the design process. This is arguably the most desirable benefit of the GAL family generic architecture approach.

An often overlooked benefit of the generic architecture approach is the added flexibility of moving functions from pin to pin after a design is completed, to simplify the board-layout process. This feature becomes evident when exploring the different ways a designer might lay out a board when a 16R4 architecture is used. Refer to the PAL16R4 logic symbol of Figure 1. There are 4 registered outputs (pins 14-17) and 4 combinational outputs (pins 12,13,18,19). This type of device gives the designer the flexibility of moving registers around between pins 14 and 17 and likewise moving combinational outputs around among pins 12,13,18,19. While there is some flexibility with this approach, the situation changes dramatically when the GAL16V8 is used.

With a GAL16V8 emulating a PAL16R4 architecture, a total of 70 different arrangements of outputs is possible while performing a specified function. Every logic symbol in Figure 2 depicts the functional equivalent of a PAL16R4, yet each output arrangement is different. Even though the architecture has already been determined, many alternative output arrangements can still be chosen, depending on the board requirements.

The ability to move any function to any pin not only greatly simplifies the board-layout procedure, but can eliminate a level of signal-routing on a multilayer board,

Figure 1. PAL16R4 Icon



as well. The fixed output placement of PAL devices often results in outputs signals having to cross on the board, requiring additional routing levels — which means additional cost. The inherent flexibility of the GAL family permits any function on any pin, thereby removing the need to cross signals externally and ultimately reducing the overall complexity of the PC board.

When the entire system-development process is taken into account — including design and layout — the user would prefer the tool that offers maximum flexibility. It is certainly advantageous to use one device that replaces virtually all of the common PLD architectures

available. GAL devices meet those requirements. Most users would also wish to be free of the constraints of predetermined, fixed architectures and one-time programmable devices. GAL devices solve these problems too. Finally, the user would like to simplify the board-layout process, with the option of moving any function to any pin, even after the architecture has been determined. With the generic architecture approach, the GAL family removes the constraints of predetermined pinout, as well. The designer is most efficient when the only constraint is his own creativity; GAL devices offer that luxury. □

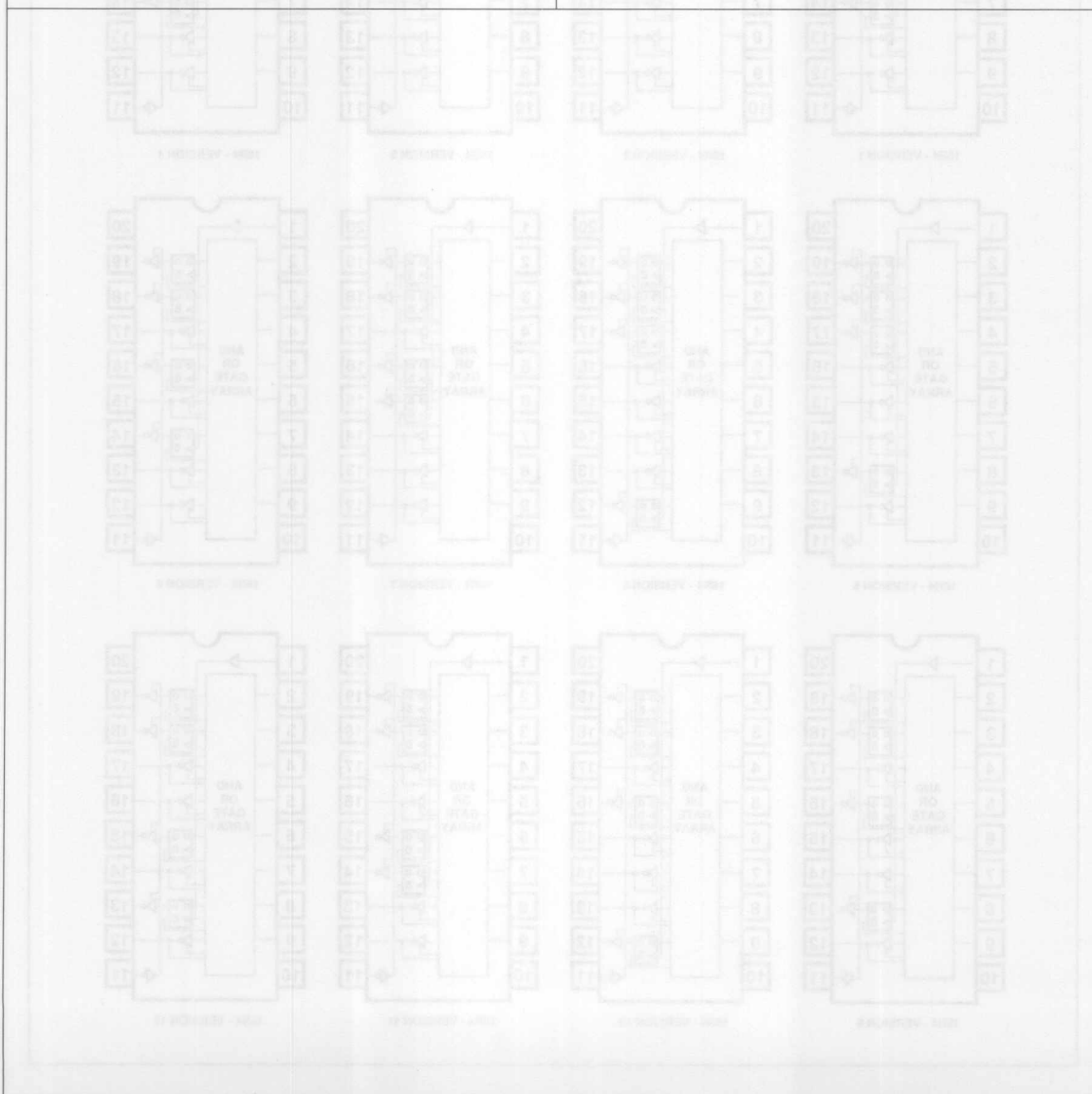


Figure 2. GAL Configurations of the 16R4

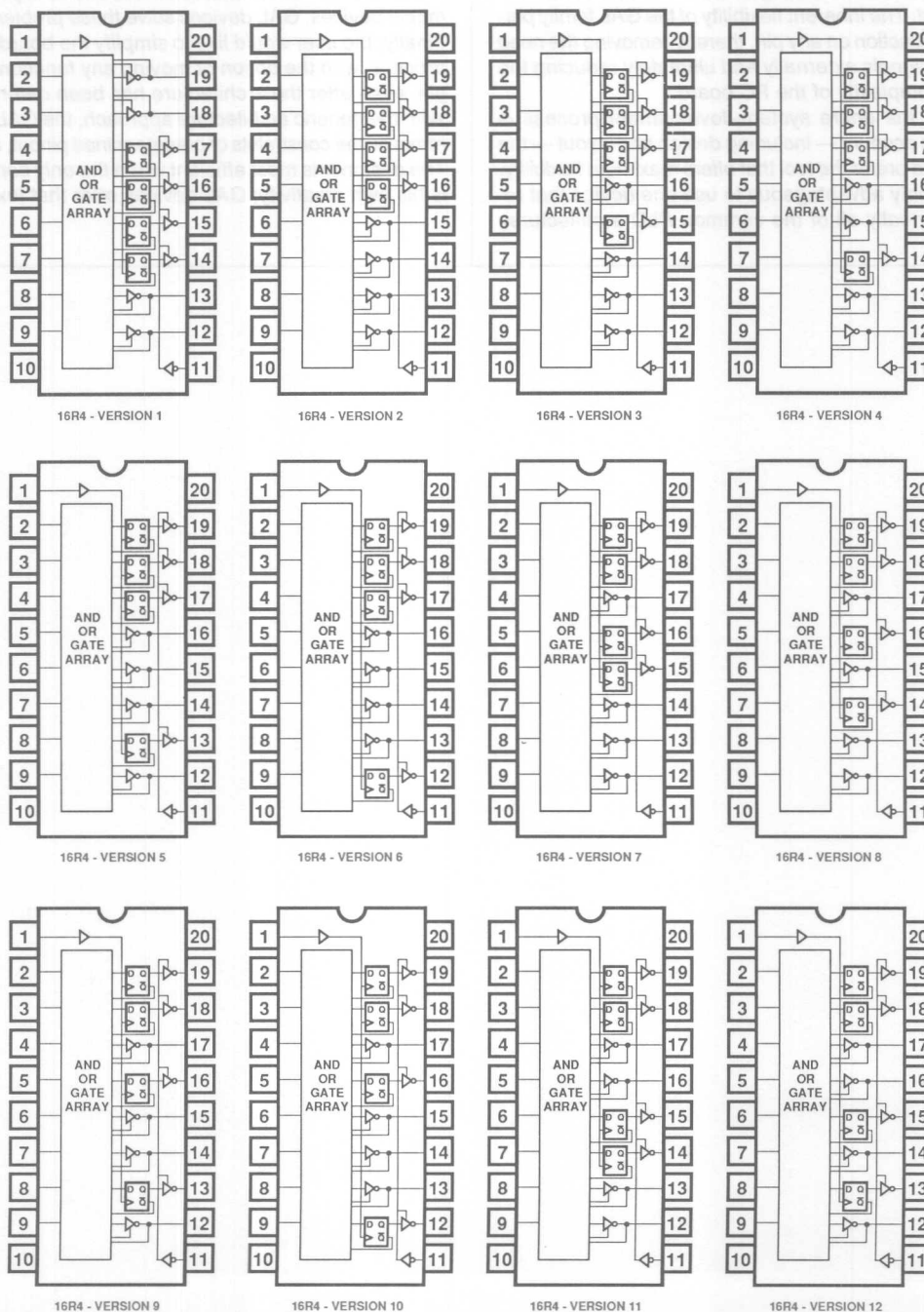




Figure 2. (cont'd)

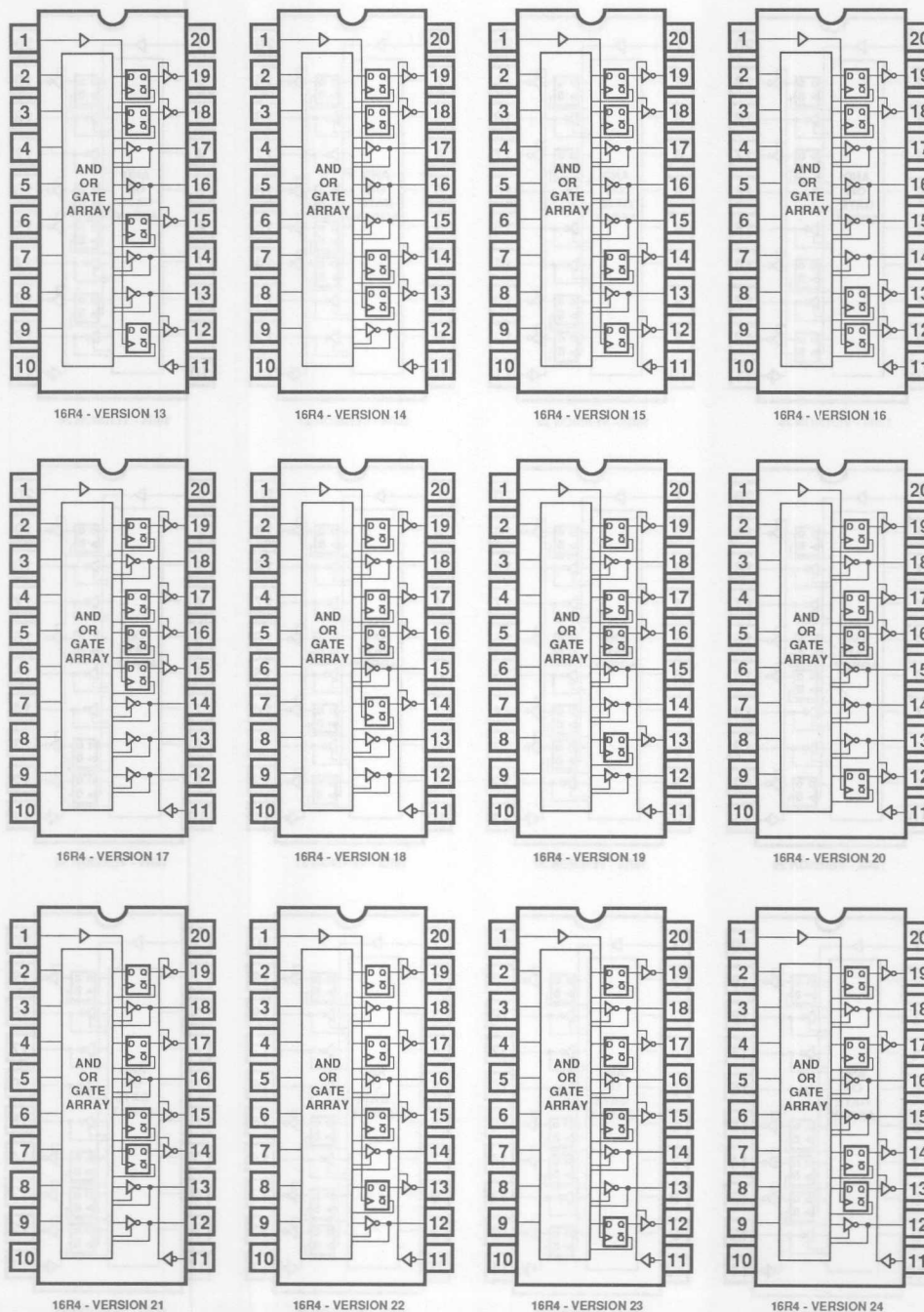




Figure 2. (cont'd)

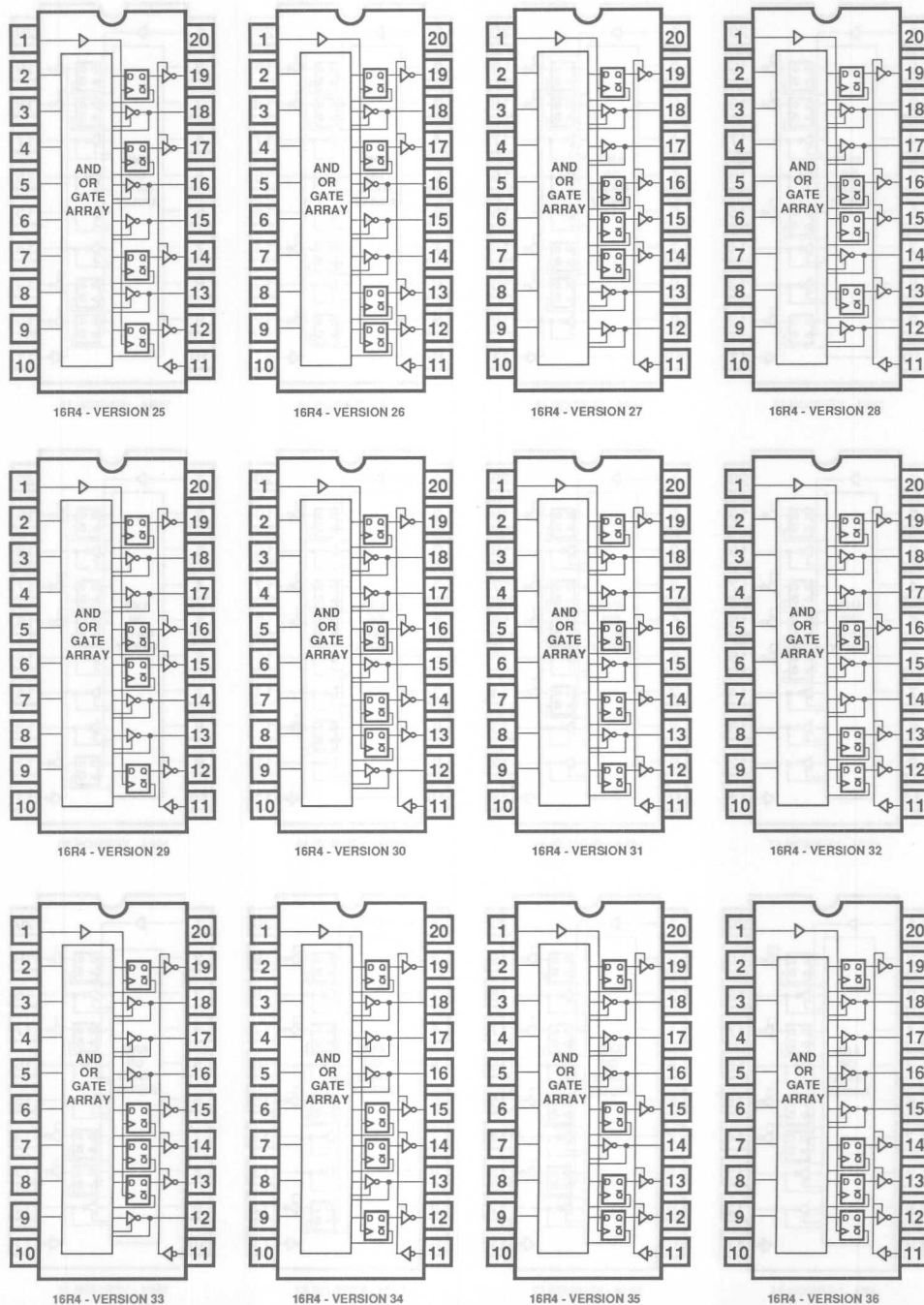
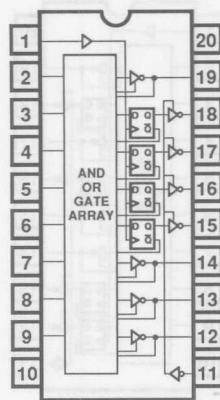
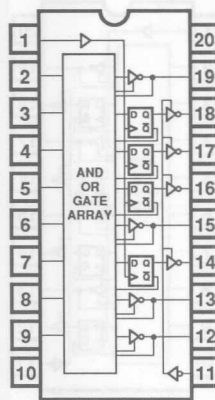


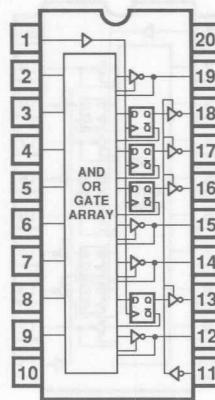
Figure 2. (cont'd)



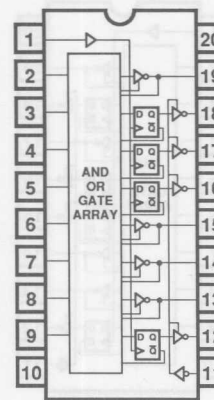
16R4 - VERSION 37



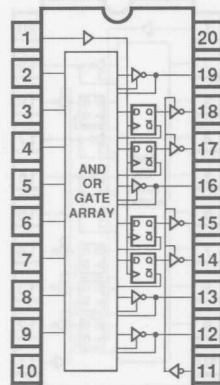
16R4 - VERSION 38



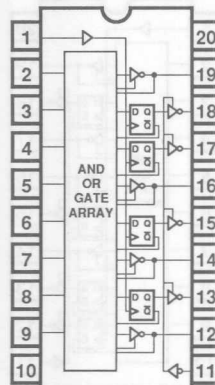
16R4 - VERSION 39



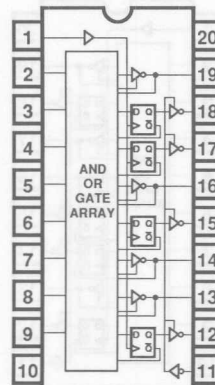
16R4 - VERSION 40



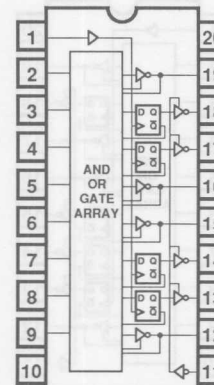
16R4 - VERSION 41



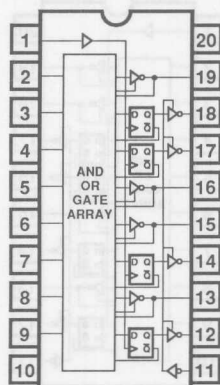
16R4 - VERSION 42



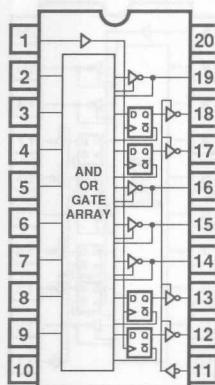
16R4 - VERSION 43



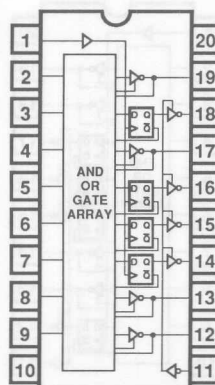
16R4 - VERSION 44



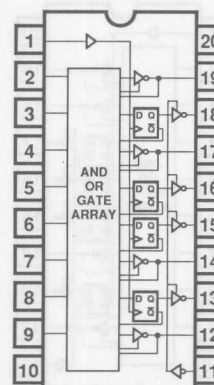
16R4 - VERSION 45



16R4 - VERSION 46



16R4 - VERSION 47



16R4 - VERSION 48

Figure 2. (cont'd)

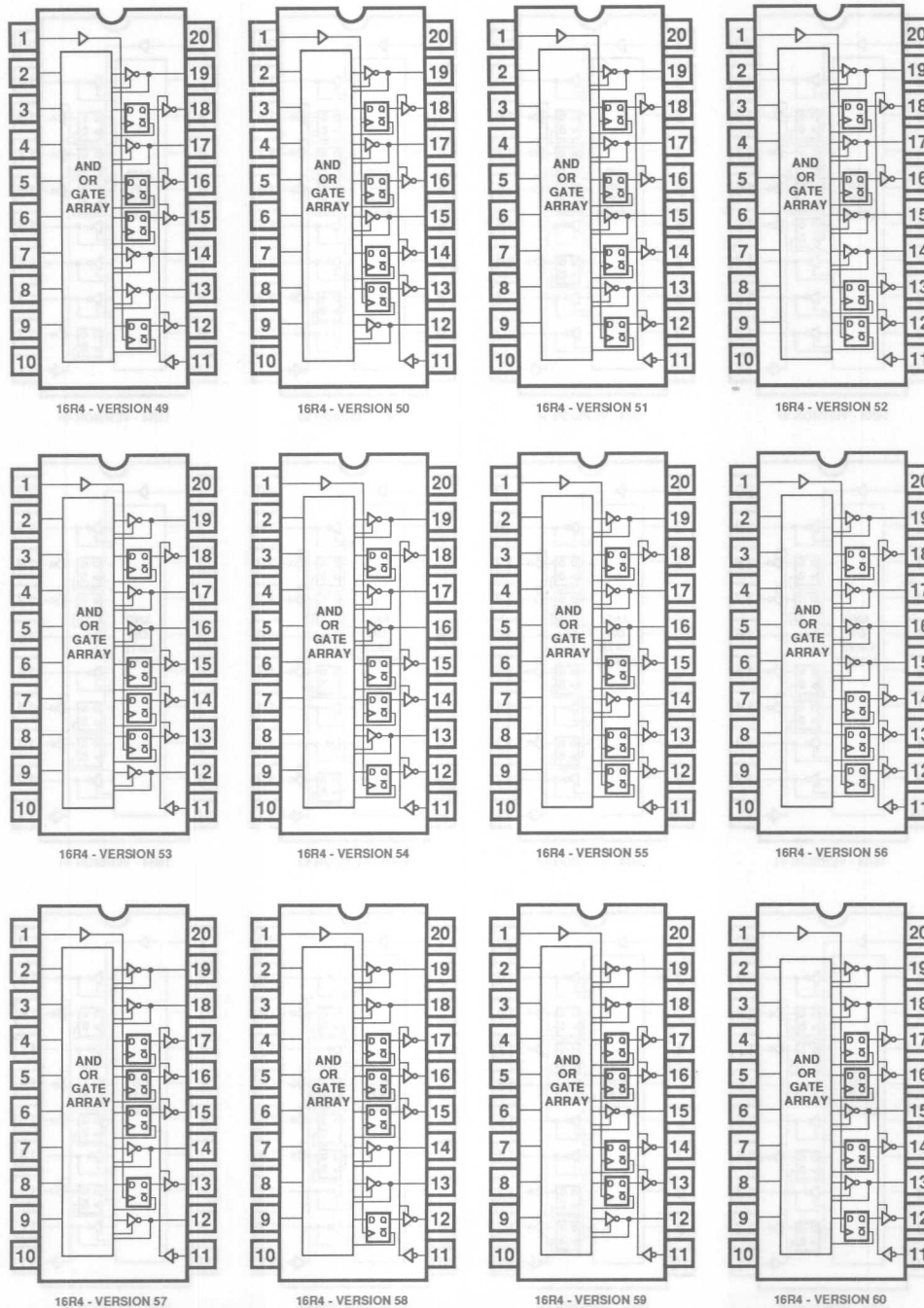
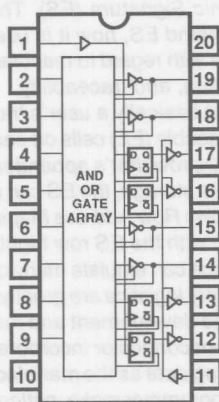
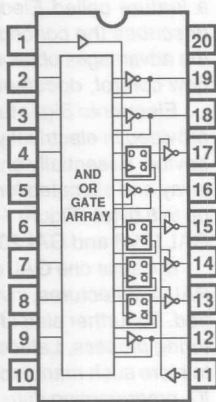


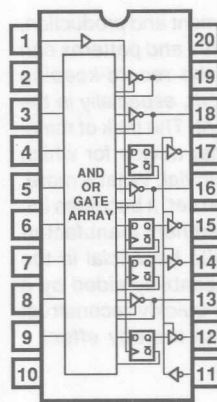
Figure 2. (cont'd)



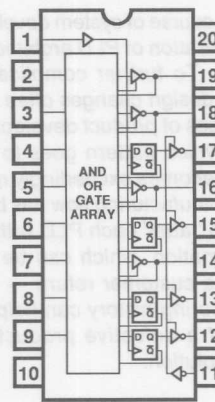
16R4 - VERSION 61



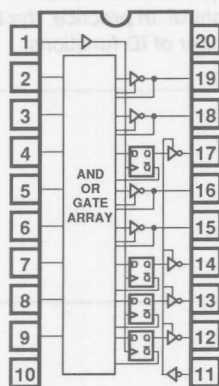
16R4 - VERSION 62



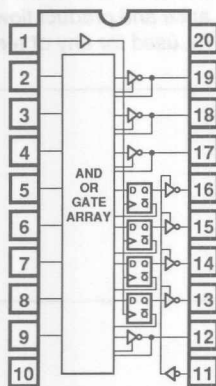
16R4 - VERSION 63



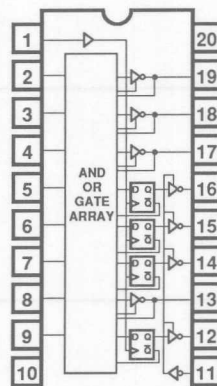
16R4 - VERSION 64



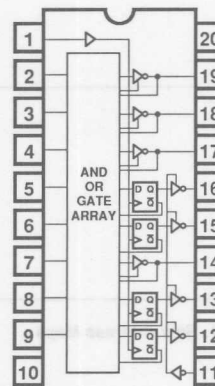
16R4 - VERSION 65



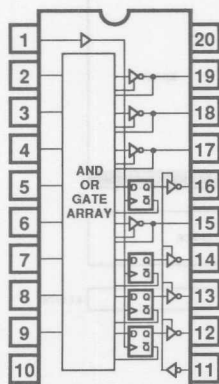
16R4 - VERSION 66



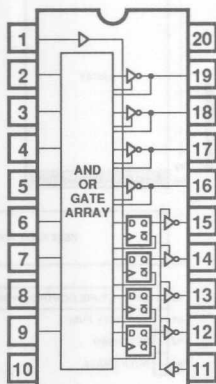
16R4 - VERSION 67



16R4 - VERSION 68



16R4 - VERSION 69



16R4 - VERSION 70

## INTRODUCTION

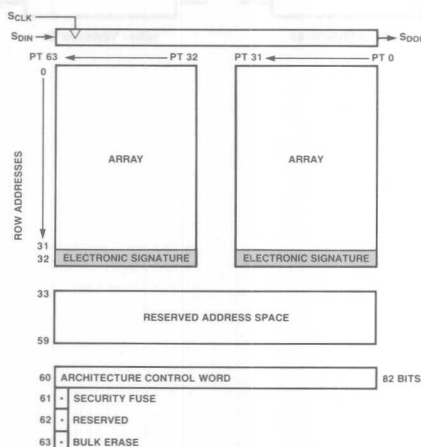
In the course of system development and production, the proliferation of PLD architectures and patterns can be great. To further complicate the record-keeping process, design changes often occur, especially in the early stages of product development. The task of maintaining 'which pattern goes to what device for which socket' becomes exceedingly nontrivial. What's more, once a manufacturing flow has been set, it becomes important to 'label' each PLD with pertinent manufacturing information, which can be quite beneficial in the event of a customer return — traceability aided by a manufacturing history can help to quickly reconstruct details of a defective product and thereby effect a speedy solution.

The Lattice GAL family can ease the problems associated with document control and traceability, thanks to a feature called Electronic Signature (ES). This brief describes the concept behind ES, how it is used, and the advantages obtainable with regard to manufacturing-flow control, documentation, and traceability.

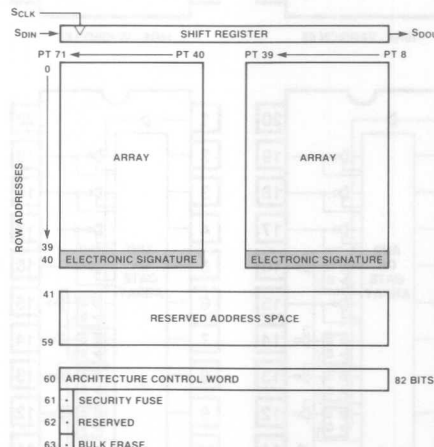
Electronic Signature is basically a user's 'notepad' provided in electrically erasable (EE) cells on each GAL device. Essentially an extra row that's appended to the array and allocated for data storage, the ES can contain up to 8 bytes. Figure 1 shows Row Address Maps for the GAL16V8 and GAL20V8, with the ES row highlighted.

Because one GAL device can emulate many different PAL architectures, inventory logistics are greatly simplified. To further simplify the development and manufacturing process, Lattice Semiconductor incorporated ES to store such manufacturing data as the manufacturer's ID, programming date, programmer make, pattern code, revision number, and product flow. The intent was to assist users with the complex chore of record maintenance and product flow control. In practice, the ES can be used for any of a number of ID functions.

Figure 1. Row Address Maps



GAL16V8



GAL20V8



For user simplicity, ES will be supported by all GAL family software support packages such as CUPL and ABEL, which are discussed with greater detail in Section 4 of this handbook. The user will be able to define data fields, specify information, and write or read ES data from each GAL device. The following paragraphs describe how ES may be managed.

Within the 64 bits (eight bytes) available for ES data storage, users may find it helpful to define specific fields, to make better use of information storage. A field may use only one bit (or all 64), and may contain a variety of topics. Some fields should probably be reserved for future expansion. The possibilities for fields are endless, and completely up to the user. As an example, Figure 2 divides the ES into five fields: manufacturer's ID, device program date, programmer ID code, pattern code, and a reserved section.

Even with the GAL device's security feature enabled, the ES can still be read. If a pattern code were stored in the ES, the user could always identify which pattern had been used in a given device. In this way, a device pattern could be confidentially retrieved. As a second safety feature, when a GAL device is erased and repatterned, the ES row is automatically erased. This prevents any situation in which an old ES might be fitted with a new pattern. (No information is better than wrong information.) It is the user's responsibility to update the ES when reprogramming.

Programming the ES is accomplished in the same manner as any other array write operation. With the GAL device in Edit mode, the ES can be selected using a row address of 32 on the GAL16V8, or 40 on the GAL20V8. By following the shift register and program timings (specified in the data sheets of Section 2) to load, program and verify the ES, a routine can be developed on most ATE

systems to pattern a user-specific signature into the device.

Though provided to assist the designers and manufacturers who utilize GAL products, making use of ES is not essential to enjoying the many benefits of GAL devices. For those willing to invest in it, however, the reduction of 'hidden costs' associated with PLDs can be significant. The following outlines some of the opportunities presented; the reader is referred to the brief on page 6-33, 'Hidden Costs in PLD Usage,' to ascribe the value of each benefit.

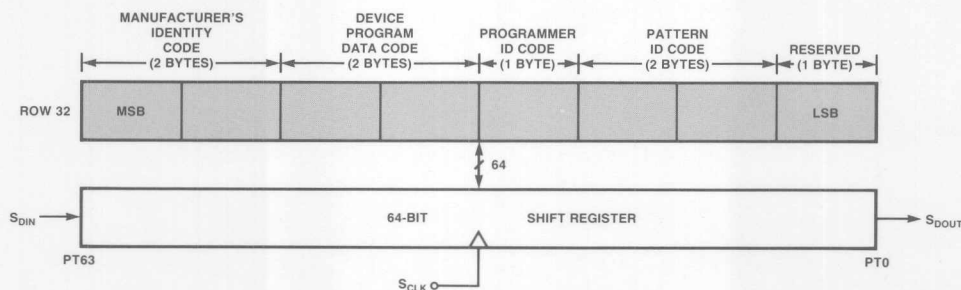
### Eliminating Labels

By automatically storing the appropriate identification information into GAL device ES locations while the programming hardware is patterning the device, the need for a costly additional handling step to apply messy gummed labels or ink is eliminated. What's more, throughput and quality of the patterned devices is greatly increased.

### Document Control

The job of document control becomes more manageable when using the GAL device ES, since a pattern code in the ES can specify each pattern and its application. This proves an absolute boon in Military programs, where accurate documentation is essential. If a change occurs, it is easily handled with a new pattern code. In fact, with a pattern code in each device, a readout can actually be conducted during board assembly. Code verification would ensure the use of properly patterned devices and serve as a quality-monitor step. Moreover, validation is simplified when checking against a lot- or board-traveler, since master devices are not required.

Figure 2. Typical ES Field Definitions





## Software Revisions

With ES, a software ID code can be stored and referenced in Document Control to a current-pattern version. When a revision occurs, a new pattern code is simultaneously stored in the ES. For the first time in PLD history, pattern codes can be monitored to verify that incorrect versions of software are not inadvertently being used. With GAL devices, of course, any material flagged with an improper pattern code can simply be sent back and reprogrammed to the current-pattern revision. Also, when security is enabled, an ES-resident pattern ID code is the only certain means of documenting which pattern resides within a device.

## Manufacturing Information

As described earlier, manufacturing information stored in the GAL device ES can help track down problems, should products be returned. If each board-assembly

location were coded into GAL devices used at that assembly site, for example, customer board returns might be linked to a common source. Also, identification codes would eliminate the need to use external labels or stamps to signify different vendors.

## Manufacturing Flow

With ES, devices can all be preprogrammed at one location and given a destination code. Upon shipment and receipt, sample readouts of destination codes could be performed to ensure that the proper devices were received.

As systems become more complex, production- and document-control costs can become dominant. Electronic Signature is one of many valuable ease-of-use features offered in the GAL device family that can tame such costs. Lattice Semiconductor will continue to deliver outstanding user support, by making the ES feature available on all GAL devices. □

## INTRODUCTION

Register preload is a testability feature that is rapidly gaining prominence throughout the chip-design marketplace. Because it allows any arbitrary state value to be loaded into a PLD's output register, this powerful feature is capable of breaking down complex logic designs into simple testable blocks. All GAL family devices from Lattice Semiconductor incorporate the feature, and though extremely valuable for testing the device during manufacture, register preload allows the discerning user to convince himself of this bold claim: because of the testability advantages of E<sup>2</sup>CMOS technology, Lattice guarantees 100% programming and functional yields for GAL devices, thereby obviating the need for the user to test GAL devices after programming. Although its comprehension is not a prerequisite for GAL device usage, this technical brief provides a technical overview of register preload and how it can be useful.

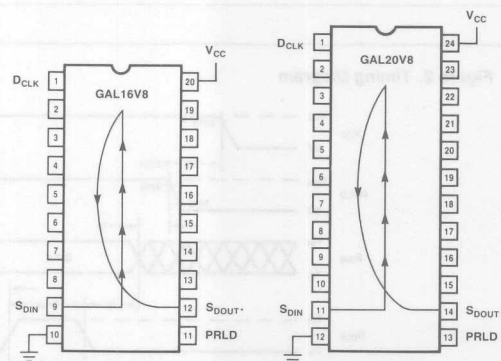
By allowing any arbitrary state value to be loaded into a device's output registers, register preload can provide an easy method of testing registered devices for logical functionality.

For conclusive testing of state-machine designs, all possible states and state transitions must be verified, not just those required in the normal machine operations. This is because, during system operation, certain events (power-up, line voltage glitches, brown-outs, etc.) can throw the logic into an illegal state. To test a design for proper handling of such conditions, a way must be provided to break the feedback paths, and force any desired — even illegal — state into the registers. Then the machine can be sequenced, and the outputs tested for correct next-state conditions.

The GAL16V8 and GAL20V8 devices include circuitry that allows each registered output to be synchronously set either high or low. Thus, any present-state condition can be forced for test sequencing. Figure 1 shows the pin functions necessary to preload the registers. The register preload timing and pin voltage levels necessary to perform the function are shown in Figure 2. (See Section 2, 'GAL Device Specifications' for parametric specifications regarding register preload.) This test mode is entered by raising PRLD to V<sub>IES</sub> (a supervoltage of 15V), which enables the serial data in (S<sub>DIN</sub>) buffer and the serial data out (S<sub>DOOUT</sub>) buffer. Data is then serially shifted into the registers on each rising edge of the clock, DCLK. Only the macrocells with registered output configurations are loaded. If only 3 outputs have registers, then only 3 bits need be shifted in. The registers are loaded from the bottom up, as illustrated in Figure 1.

6

Figure 1. Output Register Preload Pin Configurations



\*The S<sub>DOOUT</sub> output buffer is an open-drain output. This pin should be terminated to V<sub>CC</sub> with a 10K resistor.

A typical functional test sequence would be to verify all possible state transitions for the device being tested. To verify these transitions requires the ability to set the state registers into an arbitrary 'present-state' value, and to set the device inputs to any arbitrary 'present-input' value. Once this is done, the state machine is then clocked into a new state or 'next state'. The next state is then checked to validate the transition from the present state. In this way any state transition can be checked.

### Shorter Test Sequences

The difficulty in getting to certain states or conditions can lead to logic-verification sequences that are either incomplete or excessively long. Long test sequences result when feedback signals from state registers combine with inputs to determine the 'next-state' values. This condition forces the state machine to go through many state transitions before it can reach the state that requires testing. Therefore, the test sequence will be mostly state-initialization and not actual testing. The test sequence can become excessively long when a state must be reentered many times to test a wide variety of input combinations.

Consider a device programmed as a seven-bit counter that, among other functions, asserts an output signal from an eighth output once the counter reaches 127, or binary 1111111. In order to test this eighth output for the proper logic level, it is necessary to cycle the counter 127 times just to set up the desired test condition. Also, if several different input combinations need testing with the seven-bit counter in this state, each set of input conditions will require the test sequence to lengthen by 128 test vectors, of which 127 vectors are for initialization only.

Register preload allows the desired 'present state' to be loaded into the device in one test vector and allow

testing for the 'next state' with a second test vector. The benefits of this feature are readily apparent for shortening test sequences.

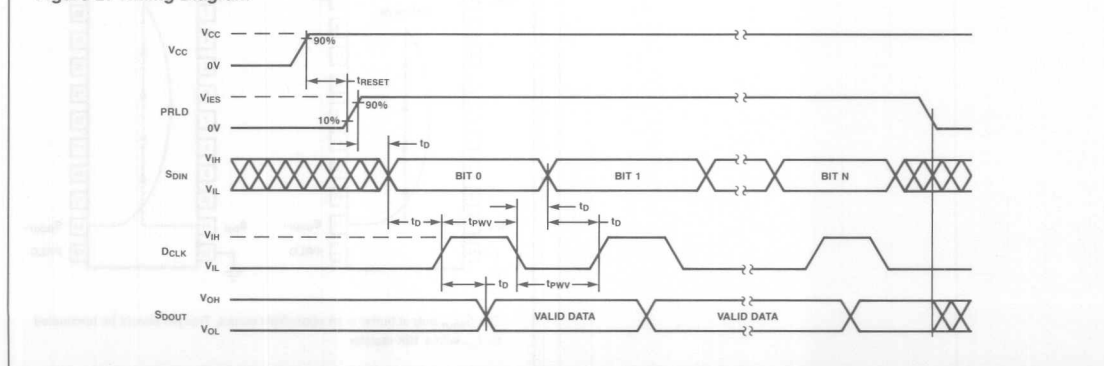
### Unreachable States

Complete logic verification is often impossible when states that need to be tested can not be entered with normal state transitions. Included among these are 'forbidden,' 'power-up,' or 'don't care' states that are not normally entered but need to be tested to ensure that they return to the defined state.

Consider a design including a counter that counts in the sequence 0, 1, 2, 5, 0, 1, 2, 5... Under normal conditions, the state registers for this counter would never reach states '3' and '4.' However, to guarantee proper functionality, these states must be entered, to ensure the machine will reset if either of these forbidden states is ever errantly entered. Without register preload, this task is impossible, since there would be no way to force the counter into forbidden states. With register preload, the states are simply loaded into the counter, which is subsequently clocked to the next state to verify proper return to a known state and normal operation.

For the most demanding users, register preload is an invaluable feature for testing all possible state transitions in a registered device, and one that can significantly shorten test sequences, as well as allow forbidden states to be tested. These benefits translate to higher-quality systems, because of the greater degree of confidence that all components in the system function properly prior to shipment. The point for all users of GAL devices to remember is that the inherent testability of Lattice Semiconductor's E<sup>2</sup>CMOS technology removes the testing burden from the user by assuring the highest-quality programmable logic devices available: 100% programming yields, 100% functional yields — guaranteed. □

Figure 2. Timing Diagram



### INTRODUCTION

While it is the responsibility of component manufacturers to maximize product quality and reliability by testing to weed out infant mortality and thereby delivering units with a low failure rate, programmable-logic-device (PLD) manufacturers have traditionally placed the responsibility of testing on the user. Primarily because bipolar fuse-link technology does not lend itself well to testability prior to programming, PAL device consumers have been forced to bear the burden of testing.

Lattice Semiconductor has changed all that, with the introduction of GAL devices. The first PLDs to use electrically erasable E<sup>2</sup> CMOS technology, GAL devices are also the only PLDs that are completely tested before shipment. Incorporating design-for-testability features, GAL devices are reprogrammed in milliseconds and tested extensively through the actual circuitry used in device operation, rather than by means of 'shadow' arrays and 'dummy' columns that one-time-programmable PLDs resort to.

When a consumer purchases a GAL device and programs it, he can feel 100% confident that the device will adhere to all performance specifications outlined in the GAL data sheet. This can be assured because prior to shipment, each GAL device has been programmed with hundreds of worst-case patterns, cycled for endurance, and tested over temperature with every scheme of architecture.

PLDs have historically been impossible to test and characterize on a volume-production level, because complete performance testing could not be accomplished until after a device was programmed for a specific application. Only the customer could verify that performance specifications were indeed being met, by testing each device fully after programming.

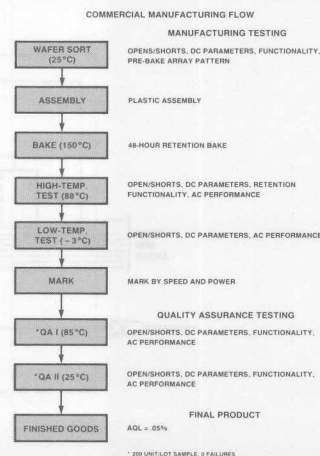
Manufacturers of bipolar fuse-link PAL devices present statistical data that implies that if the device programs, it will probably operate adequately. Consequently, the user has to program the device and test it over temperature to screen out statistical anomalies. If programming yield is 98% and functional yield is 99%, then each user can expect to return 3% of all PAL devices purchased — or three parts per hundred.

### Predictable Programming

From a standpoint of device reliability, it is important for users to distinguish between the untestable behavior of fusible links and the highly predictable and repeatable retention characteristics of the GAL device's E<sup>2</sup>CMOS arrays. Design features enable each E<sup>2</sup>CMOS cell to be checked for sufficient margin after programming, to ensure that all cells comply with the retention requirements dictated by Lattice Semiconductor's quality and reliability goals. All devices undergo retention testing, in which

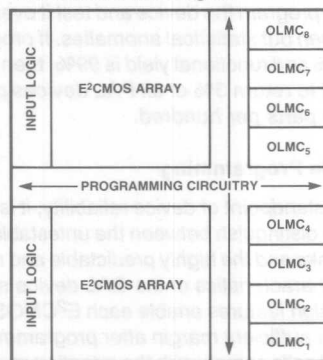
6

Figure 1. Standard Product Manufacturing Flow



every cell's charge is measured before and after a high-temperature 'bake,' to detect any unusual degradation that might portend potential failures. For the user, this means that programmer 'sensitivities' are a thing of the past. In contrast, poorly processed fuse links in a bipolar PLD — which might require a higher current to program — can combine with a 'weak programmer' (low-current-sourcing) to produce poor fusing characteristics. If proper fuse 'gapping' is not achieved 100% of the time, reliability problems will arise. With fuse-link technology, reliability can often be a function of programming hardware; with Lattice E<sup>2</sup>CMOS cells, reliability is inherent. Once programmed, the GAL device is programmed for life (unless, of course, the user decides to erase and program it again).

Figure 2. GAL Device Block Diagram



### Testing GAL Devices

The GAL device standard manufacturing flow (Figure 1) has been designed to test and stress 100% of the device circuitry over the full commercial or military temperature operating range. Only with E<sup>2</sup>CMOS technology is such an objective realizable, and Lattice is the sole PLD manufacturer able to make that claim. To thoroughly check the GAL device, its circuits are partitioned into four sections, each of which is subjected to a specific test routine. As shown in the GAL16V8 block diagram of Figure 2, the four sections are: input logic, Output Logic Macrocell (OLMC), E<sup>2</sup>CMOS cell array, and programming circuitry. The parameters and specifications of each of the four sections are thoroughly validated, as described in the following paragraphs.

Figure 3. Basic Function Test Coverage

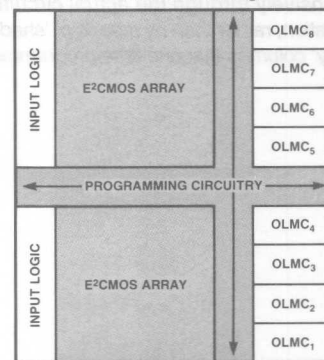
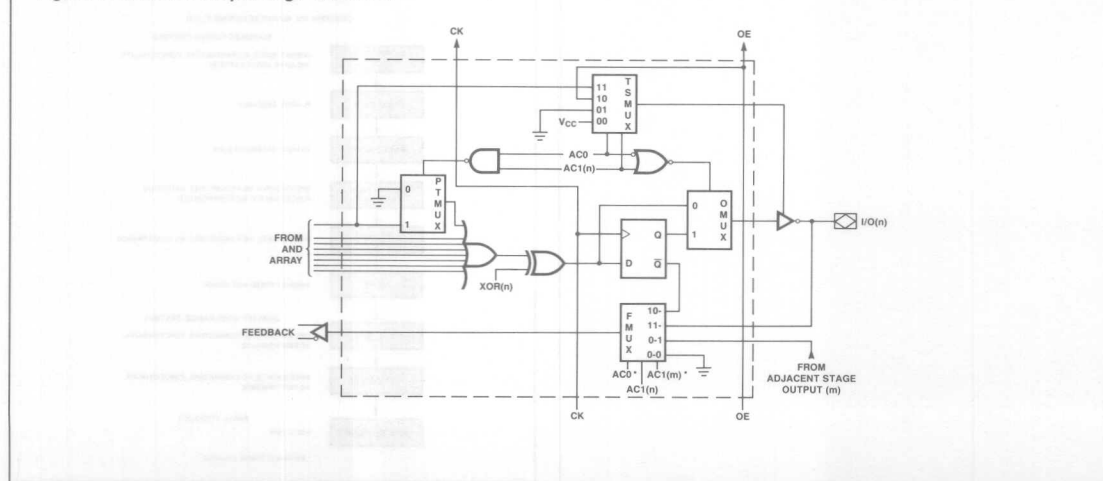


Figure 4. GAL16V8 Output Logic Macrocell





## DC Parameters

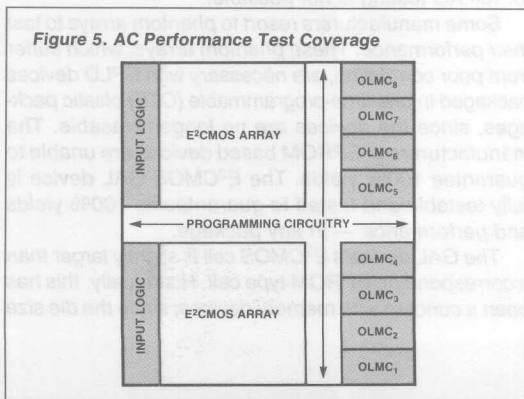
All DC parameters are measured twice, with ample guardbands: during die-probing at the wafer level, and after the GAL devices are packaged. Tests include input leakage, I/O leakage, standby current,  $V_{IL}$ ,  $V_{IH}$ ,  $V_{OL}$ ,  $I_{OL}$ ,  $I_{OH}$ , and  $V_{OH}$  — all measured under worst-case bias conditions at both temperature (either commercial or military) and voltage limits.

## Basic Function

Basic function testing verifies that the  $E^2$ CMOS cell array and all programming circuitry is operational. Timings and levels associated with serial-shift-register operation and array-addressing are used in conjunction with several data patterns to detect any array defects. Basic function testing validates that all paths through the array are continuous and programmable. During basic function testing the opportunity is also taken to test other GAL device features, such as register preload and power-on reset. The tinted region of Figure 3 indicates the areas of basic function test coverage.

## AC Performance

Since GAL device performance is taken very seriously, triple guardbands are used (voltage, timing and temperature) to ensure that device performance is well within specifications. Every timing specification is strictly tested, using worst-case patterns for each architectural configuration. By AC testing with different architectures, every AC path of the Output Logic Macrocell (Figure 4) is tested. With AC performance testing, input logic and I/O feedback are verified, as is the AC integrity of the array. Figure 5 highlights the portions of the GAL16V8 tested in this step. Note that the patterns used for AC verification are placed into the normal array; no test rows or columns are used. In fact, so extensive is the testing performed on units prior to leaving the factory that Lattice guarantees 100% AC yields.



## Active Power

For the GAL device power measurement, operating current is maximized through use of a pattern that combines standby current and peak transient power. The device is tested with an asynchronous (16L8 type) architecture to take advantage of all available outputs. Eight inputs are cycled at a frequency in excess of 15 MHz, while the supply voltage is maintained at its upper limit. (Most applications will require as little as half the power demanded by the above pattern.)

## Reliability

To optimize GAL device reliability, two special test features were incorporated into the GAL family: margin testing and internal verify. Margin testing provides the ability to individually measure the charge content of each cell, and is used primarily for two evaluations: 1) to measure each cell after programming, and thereby verify that sufficient voltage margin has been attained; and 2) to measure cell charge before and after a high-temperature stress, and thereby identify weak cells that would likely exhibit poor retention characteristics.

Internal verify provides the ability to monitor the voltage and current on internal nodes. Stress-induced changes can be identified through internal verify, and a primary use of this feature is to monitor internal circuit nodes before and after a high-voltage dynamic stress. Small changes in internal characteristics often forewarn premature circuit fatigue.

With the implementation of margin test and internal verify, reliability screens are made more effective. Used in conjunction with a dynamic and static stress, these two device features ensure dependable circuit operation and rock-solid  $E^2$ CMOS cell retention.

## Quality

To promote quality in the Lattice GAL products, strict policies and procedures for manufacturing are closely adhered to. Electrostatic-discharge (ESD) levels are continuously monitored on samples taken at each manufacturing step. 100% actual test, using generous guardbands in temperature, timings, signal levels and supply voltage, guarantees that all shipped material meets published specifications. The Lattice Quality Assurance Program, which meets or exceeds all requirements outlined in MIL-M-38510F, Appendix A ('General Specification for Microcircuits'), as well as all inspection-system requirements outlined in MIL-I-45208A, is described in detail in Section 8 of this handbook.

Testability, Quality and Reliability were foremost objectives during the design, development and manufacturing of the GAL family. The time and energy expended on this product indeed support that 100% programming yield and 100% functional yield are hard-earned facts — not just claims. □



## INTRODUCTION

The Lattice GAL 16V8 E<sup>2</sup>CMOS memory array uses the well understood Fowler-Nordheim tunneling effect for cell programming. This technology is based on a controlled-voltage, negligible-current mechanism of charge transfer. This technology was chosen for the GAL devices to enhance the flexibility and re-usability of the device by both design engineers and production personnel. The technology also allows the device to be fully tested prior to shipment using the '100% actual test' approach.

The use of the electrically erasable cells in the GAL-16V8 instead of the traditional metal fuse links offers several distinct advantages. The GAL16V8 can be (and is) fully tested prior to leaving the factory. Each and every programmable element is tested many times. There are no special test rows, columns or 'phantom' arrays necessary — the elements tested are the same elements normally programmed by the end user to implement a logic function. Therefore, the GAL device programming yield can be tested and guaranteed to be 100% with zero fallout. In contrast, the metal fuse link cannot be tested, since once the link is blown it cannot be replaced. The fuse link devices rely on test rows and columns (phantom arrays) to correlate and simulate the device's performance, so that they don't have to pattern the normal logic array. The typical fallout of bipolar product to programming is 1 to 3%, due to this inherent lack of fuse-link testability, poor correlations and inadequate simulations.

The same advantage holds for post-programming tests of AC and DC parameters. Only the E<sup>2</sup>CMOS technology allows the actual device signal paths to be tested prior to shipment from the factory. No simulation or correlation is necessary. GAL devices are tested for worst case performance under dozens of different configurations. The post-programming AC and DC yields of GAL devices are tested and guaranteed to be 100% with zero fallout.

## E<sup>2</sup>CMOS Testability Exceeds UV Devices

The E<sup>2</sup>CMOS testing approach is superior to that of the EPROM-based EPLD (UV Erasable PLD). While the EPLD devices offer the ability to be erased, this feature is impaired with a time-consuming and cumbersome process of exposure to UV light. The manufacturers of devices using the EPROM technology for their memory array are limited to one programming cycle per test insertion, because of the time necessary to erase the devices. Cell endurance and the multiple patterning necessary for full AC testing is not possible.

Some manufacturers resort to phantom arrays to test their performance. These phantom arrays, which suffer from poor correlation, are necessary with EPLD devices packaged in one-time-programmable (OTP) plastic packages, since the devices are no longer erasable. The manufacturers of EPROM based devices are unable to guarantee 100% yields. The E<sup>2</sup>CMOS GAL device is fully testable and tested to guarantee its 100% yields and performance — in any package.

The GAL device's E<sup>2</sup>CMOS cell is slightly larger than a corresponding EPROM type cell. Historically, this has been a concern with memory devices; since the die size

was array limited; however, programmable logic devices tend to not be array limited. In fact, the GAL device array occupies only 6-7% of the total die size (Figure 1). The impact of the slightly larger cell size on total die area is only 2-3% when compared to the EPROM approach. The slight cost impact of a 2% larger die area is more than offset by the ability to screen for 100% yields and the savings of being able to reject non-functional die early in the manufacturing process.

#### Data Protection

The GAL16V8 devices must be programmed by dedicated device programmers, such as those made by Data I/O, Stag, and others. The programming hardware provides an external reference voltage of slightly over 15 volts on certain pins, followed by specific timing signals on other pins, to alter the array content. As such, GAL devices are immune to false writes during power up, power down, 'brownouts,' system glitches or any other unpredictable system behavior.

The E<sup>2</sup>CMOS cell offers erase times of less than 50 ms, which takes place during programming, in a fashion that's transparent to the user. Advanced circuit design techniques implement a high-speed programming scheme that results in a full array programming time, including a bulk erase, of less than 400 ms. The programming of GAL devices is about 50% faster than comparable bipolar PAL devices, and some 20 times faster than the current tens of seconds necessary to pattern EPLD products. Faster programming times, 100% fully tested devices and 100% guaranteed programm-

ing yield (zero loss) on the GAL devices results in increased throughput and a reduced overall cost to the user.

#### Instant Error Recovery

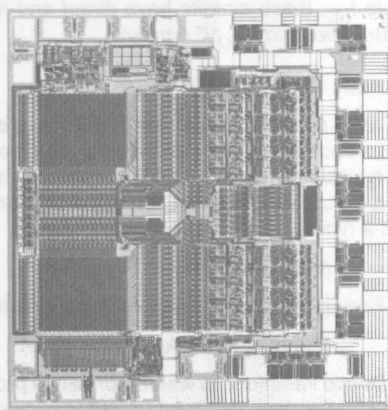
The E<sup>2</sup>CMOS process offers instant erasure in both the engineering and production environments. In the production environment, any GAL devices accidentally programmed to an incorrect or obsolete pattern can be re-programmed instantly, instead of thrown away. This minimizes the cost of production errors or changes. The 100% 'fuse' map compatibility of GAL devices with the common PAL devices allows the introduction of GAL devices into the production environment without any engineering overlay to modify patterns, files or master devices.

In the design environment, the re-useable E<sup>2</sup>CMOS device allows the designer to experiment and iterate without paying the penalty of throwing fuse-link devices away or waiting 20-plus minutes to erase UV-type devices. The designer is freed from the burden of mistakes while prototyping — a repatterned device is less than a second away. The designer can also 'build' on his devices by debugging portions of the logic, enabling additional functionality, debugging, and so on. This can greatly simplify the prototyping process.

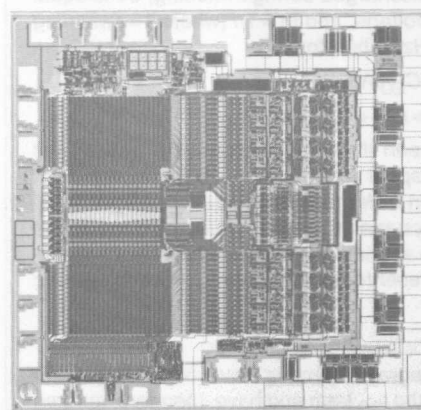
The manufacturing engineer benefits from the flexibility of the GAL macrocell configuration. Any function can be shifted to any pin, separating the chore of prototyping from the task of laying out a circuit board. The swapping of output pins can eliminate the need for cross-over traces or even an additional board level, resulting in increased reliability and reduced cost. □

6

Figure 1. GAL Device Microphotographs



GAL16V8



GAL20V8

## INTRODUCTION

Beginning with an understanding of some fundamental power-consumption characteristics of bipolar and CMOS circuit technologies, a technique can be developed that allows system designers to accurately predict typical and worst-case power consumption of GAL devices in a specific application. Here, we investigate the components of GAL device power consumption — and its dependence on the programming pattern — to develop such a structured technique.

Power consumption in integrated circuits comprises of two fundamental components: A static (DC) component and a dynamic (AC) component. The total power consumed by an integrated circuit in a system is simply the sum of these two components:

$$\begin{aligned} P_D &= P_{DAC} + P_{DDC} \\ &= (V \cdot I_{AC}) + (V \cdot I_{DC}) \\ &= V (I_{AC} + I_{DC}) \end{aligned}$$

where:  $P_D$  is the total power dissipation,  $I_{AC}$  is the dynamic (switching) current consumption,  $I_{DC}$  is the static (standby) current consumption, and  $V$  is the operating supply voltage.

Dividing both sides by the operating voltage ( $V$ ) yields the total current:

$$\begin{aligned} I_{CC(TOTAL)} &= I_{AC} + I_{DC} \\ \text{The } I_{AC} \text{ term can be further simplified:} \\ I_{AC} &= \frac{dQ_{AC}}{dt} = C_{TOTAL} \frac{dV}{dt} = (C_L + C_{EQ}) \frac{dV}{dt} \end{aligned}$$

where  $C_L$  is the total output load capacitance being driven by the integrated circuit,  $C_{EQ}$  is the total equivalent capacitance presented to the system by the integrated circuit due to the internal device nodes being charged and discharged during switching,  $dV$  is equal to the supply voltage, and  $dt = 1/f$  (where  $f$  = switching frequency in Hz). Therefore:

$$\begin{aligned} I_{AC} &= (C_L + C_{EQ}) Vf \\ I_{CC(TOTAL)} &= (C_L + C_{EQ}) Vf + I_{DC} \end{aligned}$$

It can be seen that  $I_{DC}$  is a constant, while  $I_{AC}$  is linear with respect to frequency, voltage, and load capacitance. In order to simplify the following discussion, we can assume an unloaded integrated circuit and set  $C_L = 0$ . We are now left with the equivalent capacitance ( $C_{EQ}$ ) presented to the system by the integrated circuit:

$$I_{CC(TOTAL)} = C_{EQ} Vf + I_{DC}$$

Bipolar integrated circuits have historically been dominated by the  $I_{DC}$  term and have, therefore, ignored the  $I_{AC}$  term (commonly referred to as the 'CVf' term.)

With the advent of bipolar technologies capable of attaining operating frequencies of 50-100 MHz, this CVf term is no longer negligible. Of course, when considering high-performance CMOS technologies, this CVf term is often the dominant component of total  $I_{CC}$ . In fact, many CMOS devices now separate these two components out and supply separate specifications:

$$I_{SB} = I_{CC} \text{ standby (or } I_{DC}, \text{ specified in mA)}$$

$$I_{AC} = \text{'CVf' component of } I_{CC} \text{ (in mA/MHz)}$$

The intent is to allow a designer to determine the total  $I_{CC}$  of a specific IC in a specific application.

Programmable Logic Devices (both bipolar and CMOS) have complicated matters for the users and manufacturers of these devices, owing to their inherent configurability. Each unique PLD pattern may generate significantly different values for both  $I_{SB}$  and CVf power, depending on the functions and features utilized in each application. The Lattice GAL device family has been thoroughly characterized in terms of both of these components, in order to provide a simple and accurate technique for estimating total  $I_{CC}$  in a specific application.

## Separating $I_{SB}$ and $I_{AC}$ Components

Each GAL device can be divided into two distinct areas of circuitry: one that consumes DC power, and one that doesn't. The high-speed, single-ended sense amplifier section consumes DC power ( $I_{SB}$ ) to attain performance through its ability to sense and amplify internal signal swings of as little as 50mV to full-supply-level logic signals. (Additionally, the oscillator circuitry that generates the negative substrate bias contributes about 2 mA to  $I_{SB}$ .)

All remaining circuitry is 100% CMOS technology, which essentially generates no  $I_{SB}$  component. This circuitry includes input buffers, row drivers, output logic macrocells, and output drivers. While contributing negligible  $I_{SB}$ , these circuits have direct and measurable contributions to  $C_{EQ}Vf$  current.

## Standby Current ( $I_{SB}$ )

E<sup>2</sup>CMOS GAL devices employ a high-speed sense amplifier on each product term in the array. Each sense amplifier typically consumes ~560μA of standby bias current. On a GAL16V8 there are sixty-four product terms, such that a typical  $I_{SB}$  contribution is:

$$(64 \text{ PT}) (0.56 \text{ mA/PT}) = 36\text{mA}$$

Adding the 2mA for the oscillator circuit yields:

$$\text{Total } I_{SB} = 36\text{mA} + 2\text{mA} = 38\text{mA}$$

Due to variations in wafer fabrication process parameters, this 'typical' number will vary approximately ± 20%. This variation of typical  $I_{SB}$  versus process para-

metric spread at 5.25V and 25°C is illustrated in Figure 1 for both half-power and quarter-power GAL devices.

### $C_{EQ}Vf$ Current ( $I_{AC}$ )

Due to the use of  $E^2$ CMOS technology, each of the many different paths and functions available to a system designer contributes to the  $C_{EQ}Vf$  current only when actively switching.

To accurately estimate this  $C_{EQ}Vf$  current in a GAL device requires partitioning it into independent fundamental components that can be treated separately, then added together for a net contribution. There are basically three distinct circuit areas that contribute to  $C_{EQ}Vf$  current:

- 1) Switching input and I/O buffers (includes rows in the array)
- 2) Switching product terms (Columns in the array)

Figure 1. Standby Current vs. Process Parameters

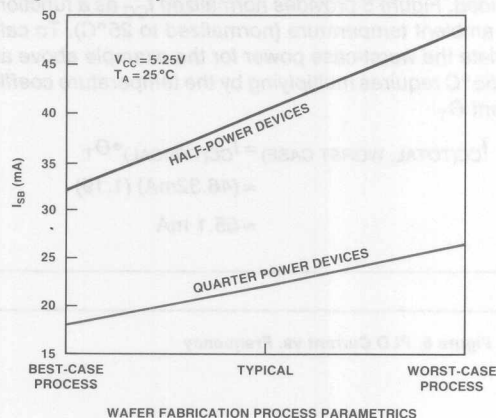
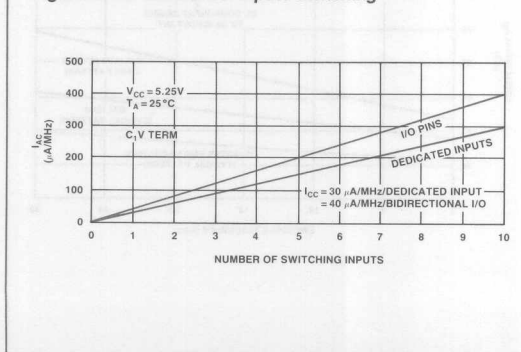


Figure 2. AC Current vs. Inputs Switching



- 3) Switching output logic macrocells and output drivers

Each of these circuit blocks represents a significant capacitance that must be charged or discharged each time it switches logic states. The  $C_{EQ}Vf$  current may be expressed as the sum of these components:

$$C_{EQ}Vf = (C_1 + C_2 + C_3) Vf \\ = (C_1V + C_2V + C_3V)f$$

Where  $C_1$ ,  $C_2$ ,  $C_3$  refer to equivalent capacitances for the above-referenced components.

Each of the three  $CVf$  components has been characterized as a function of frequency and provided for reference in Figures 2, 3, and 4. Figure 2 provides  $I_{AC}$  characteristics as a function of the number of switching-device input or I/O pins. Figure 3 illustrates the  $I_{AC}$  characteristics as a function of the number of switching product terms. Finally, Figure 4 provides  $I_{AC}$  characteristics

Figure 3. AC Current vs. Product Terms Switching

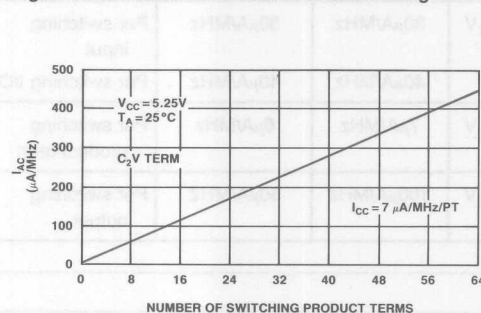
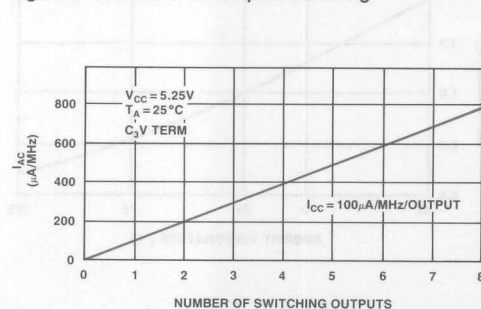


Figure 4. AC Current vs. Outputs Switching





as a function of the number of switching outputs. Table 1 provides a summary of these components.

## $I_{CC(TOTAL)}$

It now becomes a simple task to calculate total  $I_{CC}$  for a specific application by merely summing the components, according to the following equation:

$$I_{CC(TOTAL)} = I_{SB} + (C_1VI + C_2VP + C_3VO)f$$

Where  $I$  = # of switching inputs

$P$  = # of switching product terms

$O$  = # of switching outputs

To try a specific example, consider a typical 10MHz application of a half-power GAL16V8. A system applica-

Table 1. Summary of AC Current Components

Term	Half Power GAL $I_{AC}$ Contribution	Quarter Power GAL $I_{AC}$ Contribution	Description
$C_1V$	$30\mu A/MHz$	$30\mu A/MHz$	Per switching input
	$40\mu A/MHz$	$40\mu A/MHz$	Per switching I/O
$C_2V$	$7\mu A/MHz$	$6\mu A/MHz$	Per switching product term
$C_3V$	$100\mu A/MHz$	$50\mu A/MHz$	Per switching output

tion may utilize eight inputs, four outputs, two I/Os, with sixteen product terms utilized. Substituting into the equation yields:

$$\begin{aligned} I_{CC(TOTAL)} &= 38mA + [(30\mu A/MHz)(8) + (7\mu A/MHz)(16) + (100\mu A/MHz)(4) + (40\mu A/MHz)(2)]10MHz \\ &= 38mA + (240\mu A/MHz + 112\mu A/MHz + 400\mu A/MHz + 80\mu A/MHz)10MHz \\ &= 38mA + (832\mu A/MHz)10MHz \\ &= 38mA + 8.32mA \\ &= 46.32mA \end{aligned}$$

This is significantly less than the data sheet specification of 90mA. There are three considerations which explain this discrepancy:

1) This application is a 10MHz application. The data sheet specifies  $I_{CC}$  at 15MHz.

2) This application does not use all inputs and I/Os, all product terms, and all outputs. The 'CVf' component of  $I_{CC}$  is, therefore, significantly reduced.

3) The effects of temperature have not yet been included. Figure 5 provides normalized  $I_{CC}$  as a function of ambient temperature (normalized to 25°C). To calculate the worst-case power for the example above at -55°C requires multiplying by the temperature coefficient  $\theta_T$ :

$$\begin{aligned} I_{CC(TOTAL, WORST CASE)} &= I_{CC(TYPICAL)} \cdot \theta_T \\ &= (46.32mA) (1.19) \\ &= 55.1mA \end{aligned}$$

Figure 5. Normalized Current vs. Temperature

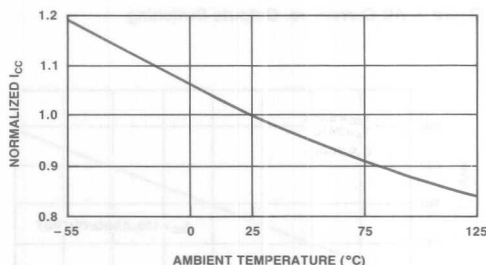
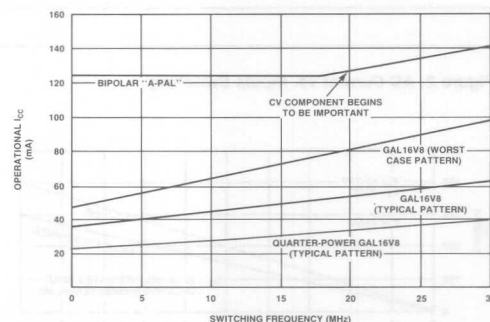


Figure 6. PLD Current vs. Frequency



Now, for illustration, look at the worst-case  $I_{CC}$  possible on a half-power GAL device. This requires:

$$T_A = -55^\circ\text{C}$$

# of switching inputs = 8

# of switching I/O = 8

# of switching PTs = 64

# of switching outputs = 8

System frequency = 15MHz

Plugging into the equation yields:

$$\begin{aligned} I_{CC(\text{TOTAL})} &= (48\text{mA} + [(30\mu\text{A}/\text{MHz})(8) + (7\mu\text{A}/\text{MHz})(64) + (100\mu\text{A}/\text{MHz})(8) + (40\mu\text{A}/\text{MHz})(8)] \\ &\quad 15\text{MHz}) 1.19 \\ &= (48\text{mA} + [240\mu\text{A}/\text{MHz} + 448\mu\text{A}/\text{MHz} + 800\mu\text{A}/\text{MHz} + 320\mu\text{A}/\text{MHz}] 15\text{MHz}) \\ &\quad 1.19 \\ &= (48\text{mA} + 27.1\text{mA}) 1.19 \\ &= (75.1\text{mA})(1.19) = 89.37\text{mA} \end{aligned}$$

This is quite close to the maximum  $I_{CC}$  of 90mA quoted on the data sheet.

It is now a simple task to generate a chart that graphically compares the total operating  $I_{CC}$  of GAL devices with that of bipolar PLDs. The following scenarios are illustrated in Figure 6:

- A) Bipolar 'A-PAL' operating  $I_{CC}$
- B) Half-power GAL16V8 with a 'typical' pattern
- C) Half-power GAL16V8 with a 'worst case' pattern
- D) Quarter-power GAL16V8 with a 'typical' pattern

Only room-temperature curves are illustrated, as both bipolar PLDs and E<sup>2</sup>CMOS GAL devices have similar temperature coefficients for  $I_{CC}$ . It can be seen that the half-power GAL16V8 significantly outperforms the bipolar PLD in terms of operating  $I_{CC}$  (by typically 90mA per device during standby, and by 75mA per device at 30MHz). □



## INTRODUCTION

The Lattice GAL family has been developed using a high-performance  $E^2$ CMOS process. CMOS processing was chosen for the GAL family to provide maximum AC performance with minimal power consumption. A drawback common to all CMOS technologies is the destructive agent, latch-up.

This brief defines the phenomenon of latch-up, how it manifests, and what techniques have been used to control it. Also described are three device features employed in the GAL family to eliminate the occurrence of latch-up, as well as the results of an intensive investigation conducted to reveal the GAL family's tolerance to latch-up.

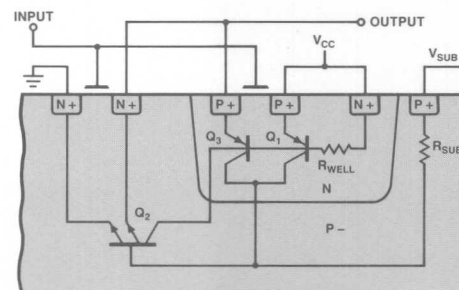
Latch-up is a destructive bipolar-device action that can potentially occur in any CMOS-processed device. It is characterized by extreme runaway supply current and consequential smoking plastic packages. Latch-up is peculiar to CMOS technology, which integrates both P and N channel transistors on one chip.

In the doping profile of a CMOS inverter, parasitic bipolar (PNPN) silicon-controlled-rectifier (SCR) structures are formed. Figure 1 shows the process cross section of a CMOS inverter, as well as the bipolar components to the parasitic SCR structure. In steady-state conditions, the SCR structure remains off. Destruction results when stray current injects into the base of either  $Q_1$  or  $Q_2$  in Figure 1. The current is amplified with regenerative feedback (assuming that the beta product of  $Q_1$  and  $Q_2$  is greater than unity), driving both  $Q_1$  and  $Q_2$  into saturation and effectively turning on the SCR structure between the device supply and ground. With the parasitic SCR on, the CMOS inverter quickly becomes a nonrecoverable short circuit; metal trace lines melt and the device becomes permanently damaged.

## Causes of Latch-Up

It has been explained that parasitic bipolar SCR structures are inherent in CMOS processing. If triggered, the SCR forms a very low-impedance path from the device supply to the substrate, resulting in the destructive event. Two conditions are necessary for the SCR to turn on: The beta product of  $Q_1$  and  $Q_2$  must be greater than unity, which, although minimized, is usually the case; and a trigger current must be present.

Figure 1. CMOS Inverter Cross-Section

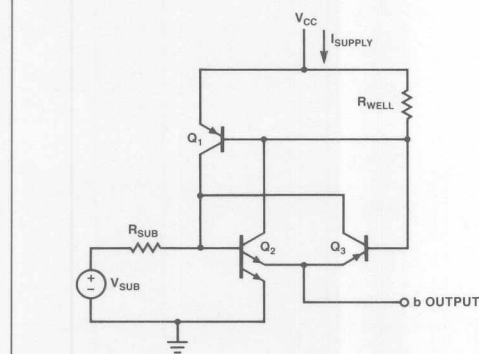


The cause of latch-up is best understood by examining the mechanisms that produce the initial injection current to trigger the SCR network. Figure 2 is a schematic of the parasitic bipolar network present in a CMOS inverter, where node "b" is the inverter output. It can be seen that two events might trigger latch-up: 1) The inverter output could overshoot the device supply, thereby turning on  $Q_3$  and injecting current directly into the base of  $Q_2$ ; and 2) the inverter output could undershoot the device ground, turning on  $Q_2$  immediately. However, a third condition could also trigger latch-up: if the supply voltage to the P+ diffusion were to rise more quickly than the N-well bias,  $Q_1$  could turn on. Within the device circuitry, overshoot and undershoot can be controlled by design. A problem area exists at the device inputs, outputs and I/Os because external conditions are not always perfect. Powering up can also be a potential problem because of unknown bias conditions that may arise.

With CMOS processing the possibility of latch-up is always present. The major causes of latch-up are understood and it is clear that if CMOS is to be used, solutions to latch-up will have to be created. As the technology evolves, solutions to latch-up are becoming more creative. Two of the more straightforward solutions are presented here.

One direct way to reduce the threat of latch-up is to inhibit  $Q_2$  (Figure 1) from turning on. This has been accomplished by grounding the substrate and reducing the magnitude of  $R_{SUB}$  through the use of wafers with a highly conductive epitaxial layer. While the technique is successful, the wafers are more expensive to manufacture, due to the extra processing required to form the epitaxial layers.

Figure 2. Parasitic SCR Schematic



The extensive use of 'guard rings' helps to collect stray currents which may inadvertently trigger an SCR structure. A disadvantage to heavy use of guard rings is the constraints placed on circuit design and topological layout, and the resulting increase in die size and cost.

### The Latch-Lock Approach

The intent of the GAL family was to implement cost-effective solutions to each major cause of latch-up. The goal was met through three device features.

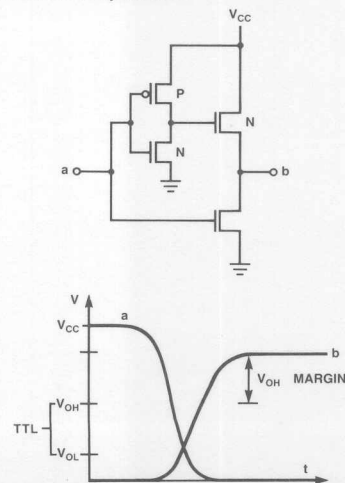
The most susceptible areas for latch-up are the device inputs, outputs and I/Os. Extreme externally applied voltages may cause a P+N junction to forward-bias, leading to latch-up. The inputs, by design, are safe; but outputs and I/Os present a danger.

To prevent latch-up by large positive swings on the device outputs or I/O pins, NMOS output drivers were used. This eliminates the possibility of turning on  $Q_3$  (Figure 2) with an output bias in excess of the device supply voltage. Figure 3 contains the effective NMOS output driver and its switching characteristics. Note that the output does not fully reach the supply voltage, but still provides adequate  $V_{OH}$  margin for TTL compatibility.

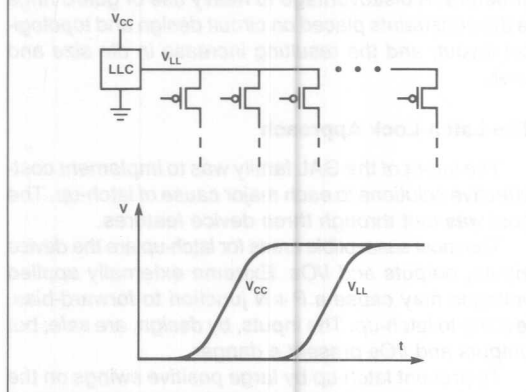
To prevent negative swings on device output and I/O pins from forward-biasing the base-emitter junction of  $Q_2$ , a substrate-bias generator was employed. By producing a  $V_{SUB}$  of approximately  $-2.5V$ , undershoot margin is increased to about  $-3V$ .

To insure that no undesired bias conditions occur with P+ diffusions, Lattice Semiconductor has developed proprietary Latch-Lock power-up circuitry, illustrated in Figure 4. In short, the drain of all P channel devices normally connected to the device supply, are

Figure 3. NMOS Output Driver



**Figure 4. Latch-Lock Power-Up Circuitry**



now connected to an alternate supply that powers up after the device N-wells have been biased and the substrate has reached its negative clamp value. This prevents any hazardous bias conditions from developing

in the power-up sequence. After power-up is complete, the Latch-Lock circuitry becomes dormant until a full power-down has occurred; this eliminates the chance of an unwanted P channel power-down during device operation.

To determine the amount of latch-up immunity achieved with the three device features utilized in the GAL family, an intensive investigation was carried out. Each step was conducted at 25° and 100°C; inputs, outputs, and I/Os were sequentially forced to -8V and +12V while the device underwent fast and slow power-ups; devices were repeatedly "hot-socket" switched with up to 7.0V.

Even under the extreme conditions specified, no instance of latch-up occurred. In attempt to provoke latch-up,  $\pm 50$  mA was forced into each output and I/O pin. The device output drivers were damaged in the battle, and still latch-up was not induced.

Based on the data, it is evident that the GAL family is completely immune to latch-up, even when subjected to a wide variety of extreme conditions, including current at inputs, outputs, and I/Os, power-supply rise time, hot-socket power-up and temperature. □

### INTRODUCTION

While the purchase price of a programmable logic device is an important consideration in identifying the most cost-effective solution for a system design, it is clearly not the only criterion. Hidden costs attributable to product testing, yield fallout, inventory management, and other factors can dramatically impact the final cost of using a PLD.

This brief investigates the overhead associated with PLD usage and the advantages of testable and reprogrammable E<sup>2</sup>CMOS GAL devices over one-time-programmable PLDs.

The GAL family of programmable logic devices is manufactured on a state-of-the-art E<sup>2</sup>CMOS process that not only provides a better speed-power product than the best bipolar devices, but offers an advantage unique among PLD manufacturers: guaranteed programming and post-programming yields of 100%.

The 100%-yield guarantee is the culmination of years of Lattice Semiconductor's circuit-design and manufacturing experience applied to the GAL device. The only way to be able to make this 100% yield statement — and to supply product that actually meets the 100% criterion — is to fully test all functions of the device, prior to shipment.

The electrically erasable (EE) matrix, unlike previous PLD matrix technologies (bipolar fuse-link and UV-erasable PROM), permits full testing of the programmability and reprogrammability of each and every matrix cell. The ability to pattern the actual matrix is extremely significant, since it also allows Lattice to test the functionality of each of the Macrocell logic blocks, under various worst-case configurations. This test approach is referred to at Lattice as 'Actual Test'. Unlike other PLD manufacturers' approaches, which include imprecise correlations, simulations, test rows, and phantom arrays, Actual Test conclusively verifies AC and DC performance of every cell in every GAL device.

### Eliminates Incoming QA

A consequence of Actual Test is that GAL devices do not require the typical incoming Quality Assurance testing that traditional fuse-link bipolar PLDs require. As such, the cost savings of using GAL devices begins the moment the parts arrive, since the average cost of an incoming QA operation — hardware, software development and maintenance, and handling — is approximately 7% of the raw device cost. Moreover, GAL devices become the optimal choice for implementation of Just-In-Time or Dock-To-Stock programs, since they eliminate the expense and time required by the incoming inspection process.

Still, a number of users require that all devices undergo incoming QA. In those cases, the use of GAL devices still simplifies the issue. A single generic test

program can be used to test all configurations of the E<sup>2</sup>CMOS-based GAL device. The expense of generating and maintaining a test program for every architecture (16L8, 16R4, 10P8, and so on) is eliminated with Generic Array Logic.

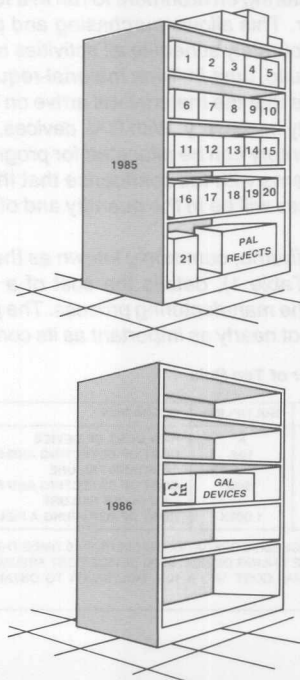
Since the QA test for fuse-link PLDs, by its nature, requires the destructive patterning of the fuse array, QA testing of bipolar PAL devices can only be done through a sample plan. At best, a sample plan can provide a crude estimate of fuse-link yield loss; moreover, sampled devices cannot be erased and must be subsequently thrown away. GAL devices, utilizing the E<sup>2</sup>CMOS process can be patterned and erased at will, allowing 100% QA of all specifications and configurations. And, the devices can, of course, be erased to allow full reuse of the sample units in manufacturing.

### Simplified Inventory Management

The generic architecture and high performance of the GAL devices allow two basic devices — the 20-pin GAL16V8 and 24-pin GAL20V8 — to directly replace approximately 70% of PLD device types currently available, including 100% of the most popular types ('L8,' 'R8,' 'R6,' and 'R4') and a sizable portion of the 'IFL' and EPLD devices. The obvious benefit of using GAL

6

Figure 1. Inventory Reduction with GAL Devices



devices is a substantial reduction in the number of part types that need be stocked (Figure 1).

Inventory management of dozens of speed-power options and device architectures is a painful process. The ideal cost of managing a device inventory adds some 2% of direct overhead; the real cost can be significantly greater, due to the risk that a shortage, 'outage,' or obsolete stock condition will exist. Improper planning could result in a shut-down of the assembly line. The generic architecture allows the GAL device to serve as insurance whenever needed to meet an immediate short-fall. The yields, at 100%, allow full planning confidence that the problem is solved.

Disposition of rejects is another inventory-management issue. The raw cost of the rejects themselves (at a 2% to 5% fallout rate) is compounded by the associated paperwork of obtaining a replacement or credit for the bad devices. Studies show that every time a buyer or purchasing agent picks up the phone or generates a debit memo, some \$30 to \$50 is spent. Followup activity — 2 or 3 calls or letters — compounds the expense. Meanwhile, the manufacturing inventory is short of devices. What's more, carrying additional 'safety-stock' as insurance against a temporary shortage results in a higher inventory-carrying cost.

### 100% Yields Reduce System Cost

Perfect yields, as provided through Actual Test, allow the manufacturing environment to run in a fully predictable manner. This allows purchasing and production-control to accurately schedule all activities and product for system build. Just-In-Time material-requisition systems assume that the material will arrive on time, in the exact quantity necessary. With GAL devices, the source product inventory can be allocated for programming to various patterns with full confidence that the final patterned devices will be in the quantity and of the quality desired.

A rule of thumb, commonly known as the 'Factor of Ten Rule' (Table 1), details the cost of a failing unit throughout the manufacturing process. The point is that unit cost is not nearly as important as its contribution to

Table 1. Factor of Ten Rule

COST*	MULTIPLIER	OPERATION
\$ 5.00	X	RAW COST OF DEVICE
\$ 50.00	10X	COST OF DETECTING AND REPAIRING A BOARD FAILURE
\$ 500.00	100X	COST OF DETECTING AND REPAIRING A SYSTEM FAILURE
\$5,000.00 +	1,000X	COST OF REPAIRING A FIELD FAILURE

\*EACH SUCCESSIVE OPERATION RESULTS IN 10 TIMES THE COST TO DETECT THE FAILING DEVICE. \$5.00 DEVICE COST ASSUMED — USE YOUR ACTUAL COST AND A 10x MULTIPLIER TO OBTAIN ACTUAL NUMBERS.

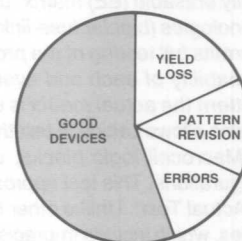
subsequent costs (or savings). The Rule basically states that the cost of detecting and replacing a defective device increases by an order of magnitude for each subsequent step of the manufacturing process.

It is extremely important to recognize that the additional difficulty and cost of using traditional PLDs has implications far beyond what the observed programming yield fallout portends. The hidden costs, time and expense aggravation of board failures (10x device cost to detect and repair), system failures (100x device cost), and the potential for field failures far outweigh the simple 2% to 5% yield losses observed on a programming fixture.

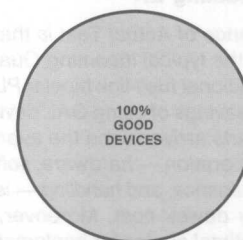
Figure 2 illustrates the differences between traditional PAL device yield loss and the 100% yields of the GAL devices. Notice that even operator errors and engineering pattern revisions are recoverable with GAL devices, which can be instantly erased and reprogrammed to the proper architecture and logic pattern.

In a typical manufacturing environment, device programming hardware patterns the array, and assuming the engineer has provided test vectors, the hardware performs a basic (slow) functional test of the device.

Figure 2. Yield Loss Comparison



TRADITIONAL PLD APPROACH



GAL APPROACH



Yield losses at these two operations average 2% to 5% and 1% to 2%, respectively.

What is not tested adequately at the PAL programming operation is the effect of partially programmed fuses that result in degraded AC performance or marginal reliability of the device. These failures are caught at board test and/or after board burn-in. Typical bipolar functional and AC parametric failure rates range between 0.5% and 2% for all manufacturers of fuse-link PAL devices. Even if one assumes the minimum failure rate of 0.5%, the system failure rates are still greatly magnified.

Two mechanisms are used to detect the failures of PAL devices: board test and system test. Using the 'Factor of Ten Rule' and assuming that board test fully screens bad devices (AC fallout), if a conservative device failure rate of 0.5% were observed, the actual parts cost would be:

$$\begin{aligned} \text{Acost} &= \text{Pcost} + (\text{Pcost} \times 10 \times 0.5\%) \\ &= \text{Pcost} + \text{Pcost} \times 0.05 \\ &= 1.05 \times \text{Pcost} \end{aligned}$$

Performing the screening at the system level, under the same scenario, makes a dramatic difference in the cost of the device:

$$\begin{aligned} \text{Acost} &= \text{Pcost} + (\text{Pcost} \times 100 \times 0.5\%) \\ &= \text{Pcost} + \text{Pcost} \times 0.5 \\ &= 1.5 \times \text{Pcost} \end{aligned}$$

These two different cost factors were determined using the conservative failure rate of 0.5%. Using the GAL

device, with its 0% failure rate, provides instead a cost factor of 1; i.e., no additional cost burden is generated.

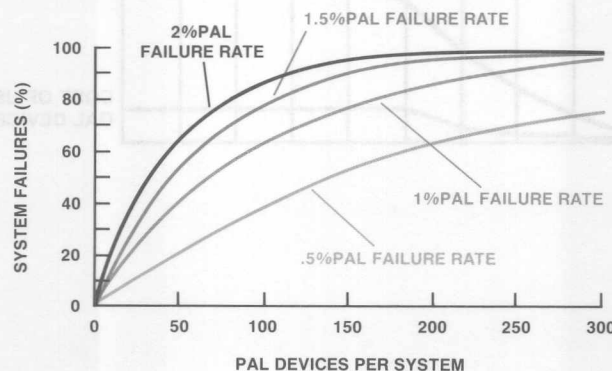
The problem caused by PLD failures obviously grows in proportion to the number of devices in a system, since the probability of a failure among a group of PAL devices is higher than that for a single device. Figure 3 plots the probability of a board or system not working, as a function of the number of devices per system, for a variety of device failure rates.

For example, at a unit failure rate of 1.0%, a system incorporating 30 PAL devices will exhibit a 25% failure rate. That means that 1 out of every 4 systems will have to be reworked, at tremendous cost. The replacement of an average 0.5% of the units in a system results in an actual 8% added to the hidden device cost.

The difficulty in replacing board failures is compounded by the removal of soldered units. It is quite easy to destroy a board with the removal and replacement of a defective device.

Systems that fail in the field are not only the most costly in terms of dollars and cents, but in customer relations, as well. They require responding rapidly and performing repairs in a less-than-ideal environment, without the complete tools and supplies available at the factory. Field failures will always occur to some degree. However, the use of GALs can help reduce field repair costs when they do occur — even if the failing device is a traditional bipolar PLD — since the generic, erasable nature of GAL devices allows a minimum of field inventory to be carried, to debug system failure problems caused by other devices. The panel on the next page provides guidelines for calculating PLD usage costs. □

Figure 3. Probability of System Failures Using Bipolar PLDs





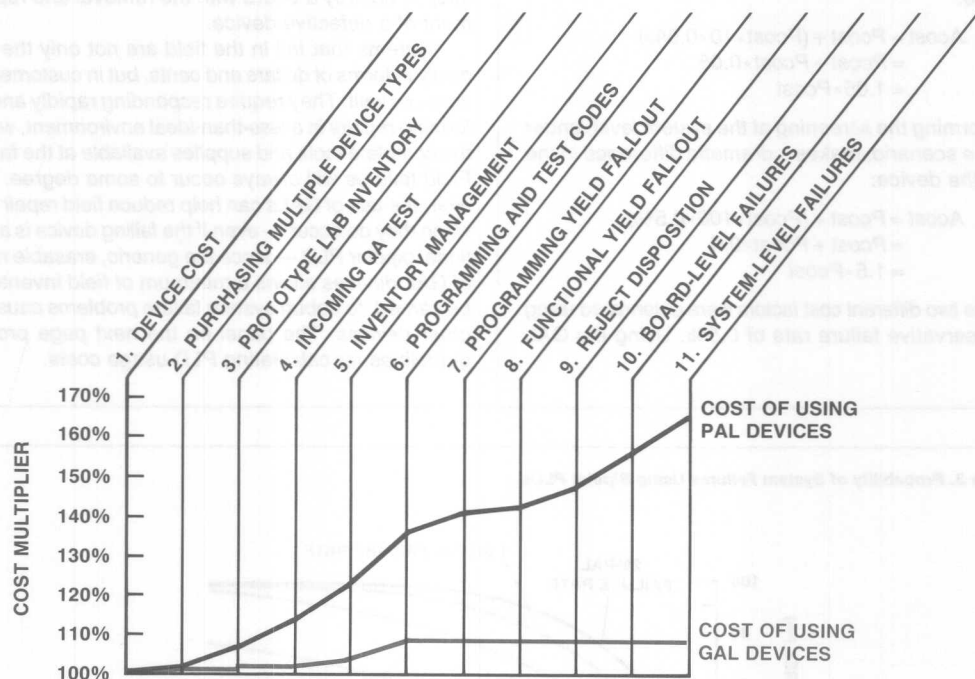
# PLD COST ANALYSIS

The cost of using a PLD goes well beyond simply the raw device cost. Programming and vector-test yields are obvious contributors to higher unit cost. The less-obvious and hidden costs tend to be much more difficult to identify and quantify.

The purpose of the costing example is to provide the basis for your own cost analysis, using your own overhead and yield numbers. Estimates for reasonable ranges of the cost contributors are shown as a guide to using your own numbers.

The explanation for each of the contributors to the device cost multiplier follow the figure. These cost multipliers include the overhead for each operation, and as a result, are higher than the observed costs.

The example shown is based on actual data from a 100,000-piece-per-year user of traditional bipolar PLDs. The environment is a typical, high-volume, quality-controlled one. The GAL device checks in at 1.09 times the normalized cost, while the actual cost of using the bipolar PLD is 1.66 times — almost 40% higher.



**1 Device cost** is normalized to unity so that the raw purchase price has no bearing on the other cost factors.

**2 Purchasing Multiple Device Types** instead of the single GAL device adds to overhead in the purchasing and receiving departments. This contributes approximately 2% as the availability, quality, and quantity issues are resolved with each order. The GAL approach reduces this number to 1.25% with inventory simplification.

**3 Prototype Lab Inventory** and usage typically adds 5% to maintain experimentation stock of multiple device types for board debug. The GAL device multiplier is 1%, since the device can be reused over and over again.

**4 Incoming QA Test** and programs cost more than may be immediately apparent, with a 7% adder. The generation and maintenance of the software and hardware for the dozens of bipolar devices is considerably more expensive than the single GAL device software required. No sample-program waste is induced. Only the aspects of handling are required for GAL devices, resulting in a reduction to 1% (or 0%, if you eliminate the incoming QA operation entirely).

**5 Inventory Management** includes shelf space, safety stock, depreciation, obsolete stock write-off and personnel to maintain adequate control of the units. A typical overhead is 10%. The simplified GAL operation involves no safety or obsolete stock and a minimum of device types, adding a maximum 2% to 3% to overhead.

**6 Programming and Test** includes all handling and hardware expenses. Inventory issuance, counting and returns, handling during the program/test operation, labels, and paperwork contribute to a 12% multiplier. The 100% yielding, generic GAL approach reduces the problem to 4%.

**7 Programming Yield Fallout** is directly observed as bad units. A typical bipolar range is 1% to 4%. GAL devices have 0% yield fallout — guaranteed.

**8 Functional Yield Fallout** is detected by the device programmer immediately after programming, through the use of test vectors, and can average 1% to 3%. GAL devices guarantee 0% functional fallout. It should be noted that using test vectors does not screen out inadequate for AC performance, which will be manifested as a board failure.

**9 Reject Disposition** overhead runs 5% to obtain replacements and credits for fuse-link devices. Zero rejects with GAL devices eliminates costs associated with reject disposition. Notice that the cumulative multiplier for only the program/test/reject of fuse-link devices is 1.10, compared with GAL devices' 1.00 multiplier.

**10 Board-Level Failures** are typically where AC failures are detected. The 'Factor of Ten Rule' exacerbates the impact of the observed 1% to 4% fallout to an overall cost impact of 7% to 10%. GAL devices exhibit no board-level fallout (and therefore no cost impact). Board throughput is also a major cost contributor, with typical reworks of 20% to 30% a consequence of PAL quality levels.

**11 System-Level Failures** add 8% to 15% to the PLD cost, taking into consideration a 100x 'Factor of Ten Rule' multiple. GAL devices again provide 100% yields, and therefore exhibit no system-level-failure cost impact. □

Programming Yield Failure is directly related to the number of devices per die. A typical die size range is 100 to 150 devices per die. A typical die size range is 100 to 150 devices per die. A typical die size range is 100 to 150 devices per die.

Functional Yield Failure is detected by the customer programmer immediately after programming. The customer programmer immediately after programming. The customer programmer immediately after programming. The customer programmer immediately after programming. The customer programmer immediately after programming.

Reject Allocation overhead runs 5% to 10% of the total cost. Reject Allocation overhead runs 5% to 10% of the total cost. Reject Allocation overhead runs 5% to 10% of the total cost. Reject Allocation overhead runs 5% to 10% of the total cost. Reject Allocation overhead runs 5% to 10% of the total cost.

Board-Level Failure is typically when the board is tested. The board is tested. The board is tested. The board is tested. The board is tested. The board is tested. The board is tested. The board is tested. The board is tested. The board is tested.

System-Level Failure adds to the total cost. System-Level Failure adds to the total cost. System-Level Failure adds to the total cost. System-Level Failure adds to the total cost. System-Level Failure adds to the total cost. System-Level Failure adds to the total cost. System-Level Failure adds to the total cost. System-Level Failure adds to the total cost. System-Level Failure adds to the total cost.

Device cost is normalized to unity so that the two purchase prices can be compared on the same basis. Device cost is normalized to unity so that the two purchase prices can be compared on the same basis. Device cost is normalized to unity so that the two purchase prices can be compared on the same basis.

Programming Multiple Device Types means that the single GAL device adds to overhead in the programming and receiving segments. This contributes approximately 5% to the available capacity and quality issues are resolved with each order. The GAL programming reduces the number of 1,000s with inventory simplification.

Prototype Lab Inventory and usage typically adds 5% to maintain equipment stock of multiple device types for board debug. The GAL device multiplicity is 10% since the device can be reused over and over again.

Incoming QA Test and program cost more than they are immediately apparent. With a 1% defect rate, the generation and maintenance of the software and hardware for the dozens of boards is considerable. The single GAL device adds only more expense than the single GAL device adds. No sample-program waste is incurred. Only the aspects of handling are required for GAL devices, resulting in a reduction in the cost of the incoming QA operation entirely.

Inventory Management includes stock, space, safety stock, depreciation, obsolete stock control and personnel to maintain adequate control of the inventory. A typical overhead is 10%. The single GAL device involves no safety or obsolete stock and a minimum of device types, adding a maximum 5% to the overhead.

Programming and Test includes all programming and hardware expenses. Inventory, hardware, and testing, handling during the normal test cycle, that, label, and paperwork contribute to a 15% overhead. The 100% testing, general GAL equipment reduces the problem to 5%.

INTRODUCTION	1
--------------	---

GAL DEVICE SPECIFICATIONS	2
---------------------------	---

LOGIC TUTORIAL	3
----------------	---

USING DEVELOPMENT TOOLS	4
-------------------------	---

GAL DEVICE APPLICATIONS	5
-------------------------	---

TECHNICAL BRIEFS	6
------------------	---

E <sup>2</sup> CMOS TECHNOLOGY OVERVIEW	7
---	---

GAL DEVICE QUALITY AND RELIABILITY	8
------------------------------------	---

ARTICLE REPRINTS	9
------------------	---

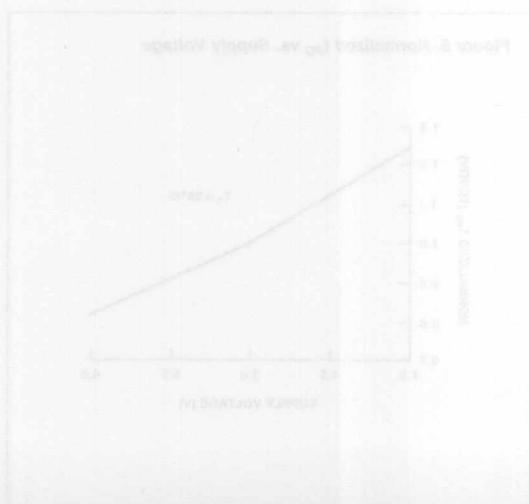
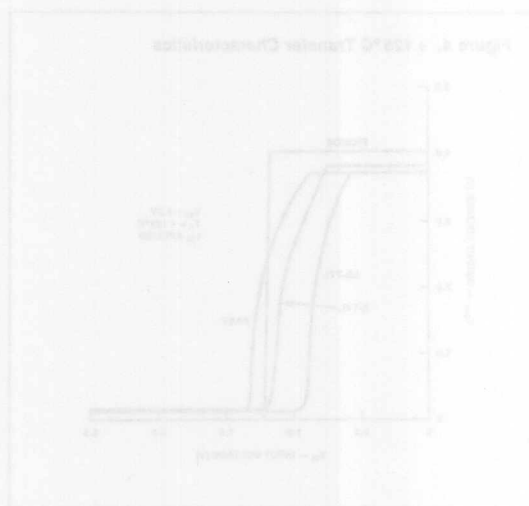
APPENDICES	10
------------	----

SALES OFFICES	11
---------------	----



## INTRODUCTION

The E<sup>2</sup>CMOS technology employed on all GAL devices combines a state-of-the-art, high-performance CMOS static RAM process with an electrically erasable, nonvolatile memory process to form Lattice Semiconductor's E<sup>2</sup>CMOS technology known as UltraMOS®. This technology yields products with the best speed-power characteristics available from any user-programmable technology.



A cross section of the Lattice E<sup>2</sup>CMOS technology is illustrated in Figure 1, with key process features listed in Table 1. The foundation of the technology is an oxide-isolated N-well CMOS process fabricated on a bulk, P-type substrate. The use of negative substrate-bias generation eliminates the requirements for expensive (and defect-inducing) epitaxial processing techniques, while enhancing device performance due to reduced junction capacitance on all N-channel devices. The oxide-isolation technique serves to further reduce device capacitance in all areas. The E<sup>2</sup>CMOS technology utilizes fully self-aligned, polysilicon-gate processing coupled with low-temperature process steps to minimize key circuit- and device-performance-limiting parasitic capacitances, such as gate-source and gate-drain overlap capacitances.

The floating-gate memory-cell technology utilized is based upon a 100-Angstrom Fowler-Nordheim tunnel-oxide technology, which was chosen due to its high quality, reliability, and repeatability. In fact, the memory cell proliferated throughout the GAL family of devices has proven its reliability through exhaustive testing (millions of write-erase cycles) and over extended periods of time (over 1,000,000 device-hours have been logged at the time of this writing).

Figure 1. E<sup>2</sup>CMOS Process Cross-Section

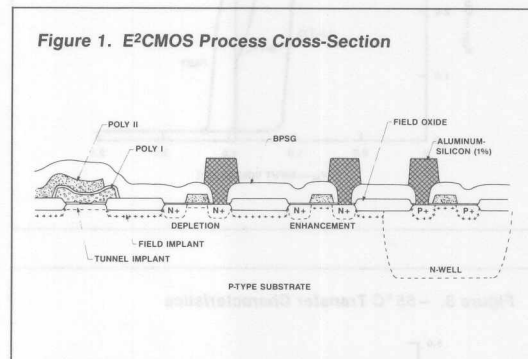


Table 1. E<sup>2</sup>CMOS Dimensional Features

UltraMOS E <sup>2</sup> CMOS Process	
P-Type Substrate (200-cm)	
N-Well Bulk Silicon CMOS	
Double-Level Polysilicon	
Single-Layer Metalization	
1.2μm Channel Length	
1.2μm Minimum Feature Size	
<b>Oxides</b>	
525 Å	First Gate
420 Å	Second Gate
580 Å	Interpoly
100 Å	Tunnel Oxide
<b>Key Pitches Utilized</b>	
5.2μm	Active Area
4.0μm	Second Poly
4.8μm	Metal



## E<sup>2</sup>CMOS Circuitry

The circuitry employed on all GAL devices has been specifically designed to provide key user benefits. AC and DC parametric performance has been optimized over the full military temperature and voltage range, through the use of substrate-bias generation coupled with temperature-compensated, self-biasing sense amplifiers.

All GAL devices have been designed to provide total parametric compatibility with existing bipolar TTL circuit technologies. Figure 2 illustrates the typical room-temperature transfer characteristics of a number of TTL circuit technologies, along with the typical GAL device characteristic. The low-temperature and high-temperature characteristics are provided in Figures 3 and 4.

Figure 2. Room Temperature Transfer Characteristics

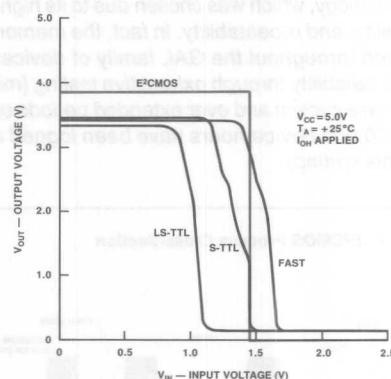
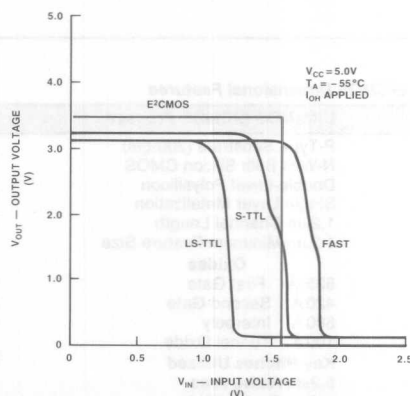


Figure 3. -55°C Transfer Characteristics



The effects of temperature and voltage on two key AC parameters (input-to-output and clock-to-output delays) can be studied in figures 5, 6, 7 and 8. The scales have been normalized to 25°C and 5.0V operation, for reference.

The advantages of E<sup>2</sup>CMOS technology become apparent upon examination of these figures. The transfer characteristic is significantly improved with E<sup>2</sup>CMOS, both in terms of static  $V_{OH}$  level (at specified  $I_{OH}$ ) as well as the 'squareness' and narrow transition region of the curve. The switching threshold is extremely stable, as compared with that of TTL circuit technologies, whose thresholds are based upon PN diode stacks; the latter has a strong exponential dependence on temperature, while E<sup>2</sup>CMOS thresholds are primarily a function of

Figure 4. +125°C Transfer Characteristics

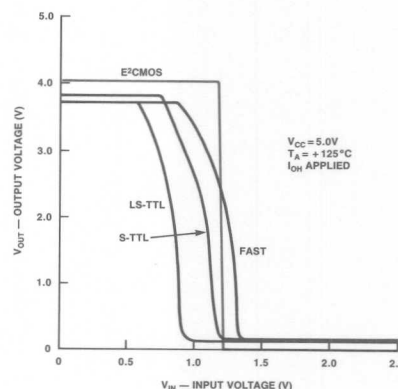
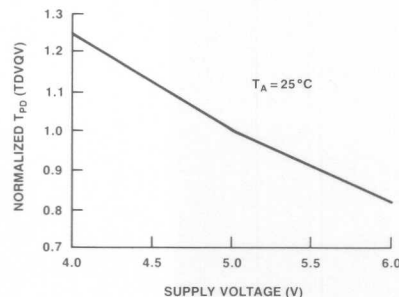


Figure 5. Normalized  $t_{PD}$  vs. Supply Voltage



device size ratios, which are, of course, not temperature-dependent.

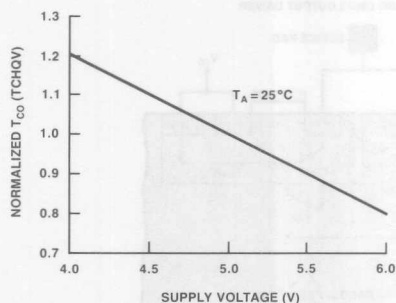
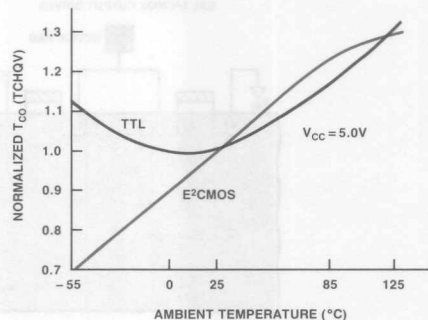
The normalized access times vs. temperature curves illustrate the linearity of E<sup>2</sup>CMOS, as contrasted with the classical bipolar 'bowl' shaped temperature characteristic. It can be seen that E<sup>2</sup>CMOS produces performance commensurate with TTL above 25°C, while significantly outperforming TTL characteristics below 25°C.

### E<sup>2</sup>CMOS Output Characteristics

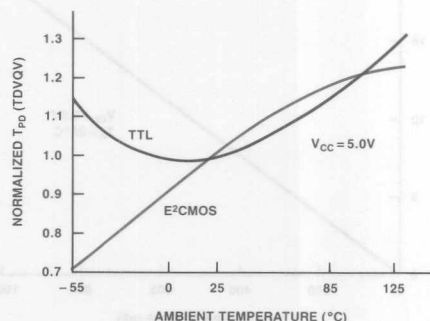
The schematic representation of the output driver utilized on all GAL devices is illustrated in Figure 9. This is a specially designed high-drive output stage with two key benefits for the system designer.

The first point to notice is the exclusive use of N-channel transistors for both pull-up and pull-down devices in the push-pull stage connected to the device pin ( $Q_1, Q_2, Q_3$ ). The use of P-channel devices on circuit outputs is one of the major sources of destructive 'SCR latch-up' common to many CMOS technologies. The reason for this is due to output overshoot, undershoot, or 'ringing' due to signal reflections and noise in a system application. When a device pin with both P- and N-channel transistors experiences such conditions, a forward-biased PN diode junction is created and the common four-layer 'PNPN' SCR structure is readily available, as illustrated in Figure 10. The Lattice GAL device output driver creates no four-layer PNPN structure and thus, eliminates potential destructive latch-up.

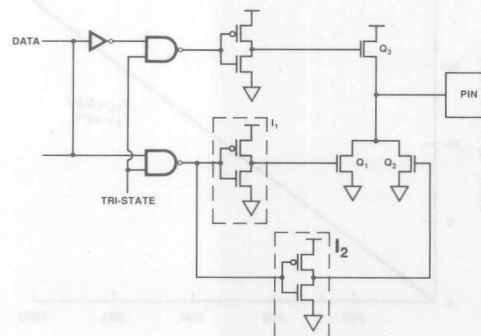
Figure 6. Normalized  $t_{CO}$  vs. Supply Voltage

Figure 8. Normalized  $t_{CO}$  vs. Temperature

**Figure 7. Normalized  $t_{PD}$  vs. Temperature**



**Figure 9. Phased Output Turn-On Circuit**



The second salient feature of the GAL output-stage design is the phased, 'soft' turn-on of its high-current (24mA) pull-down drivers,  $Q_1$  and  $Q_2$ . This feature benefits system designers by controlling transient current spikes, which often occur when switching heavy, capacitive loads and many outputs in parallel. Dynamic switching spikes often play havoc with system  $V_{SS}$  and  $V_{CC}$  bus lines, due to inductive coupling according to the equation:

$$V_{NOISE} = L_{BUS} \left( \frac{di}{dt} \right)$$

where  $V_{NOISE}$  is the equivalent noise voltage, and  $L_{BUS}$  is the bus inductance.

The phased output turn-on circuit serves to limit the  $V_{NOISE}$  component by effectively reducing the  $di/dt$  term. This is accomplished through the use of paired pull-down devices (Figure 9). In effect, the circuit has two output drivers, a dynamic driver ( $Q_1$ ) and a static driver ( $Q_2$ ).  $Q_1$  is driven by a very fast inverter stage ( $I_1$ ), while  $Q_2$  is driven by an inverter stage ( $I_2$ ) designed to turn on more slowly (by about 1.5ns), but still turn off quickly. This design effectively smoothes the dynamic output-current spike generated when switching heavy, capacitive loads, yet provides a solid DC  $I_{OL}$  drive characteristic.

The propagation delay induced by capacitive output loading is illustrated in Figures 11 and 12. The DC out-

Figure 10. CMOS Latch-Up Scenarios

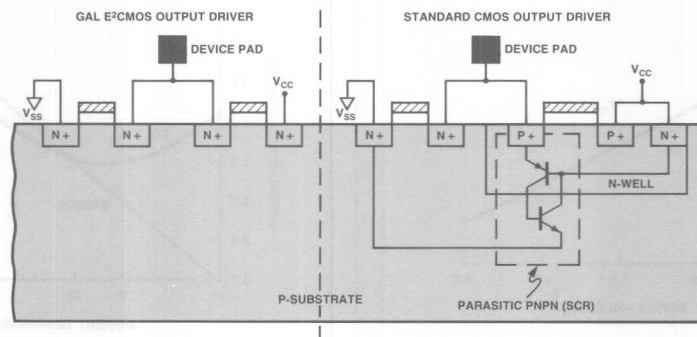


Figure 11. Delta  $t_{PD}$  vs. Output Loading

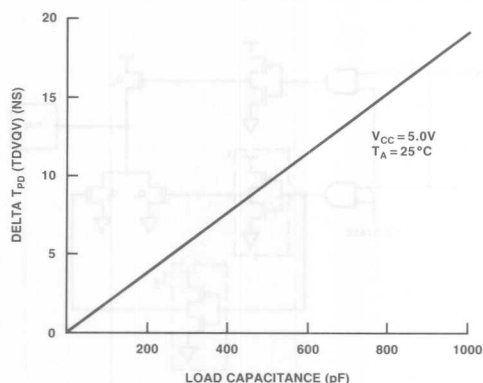
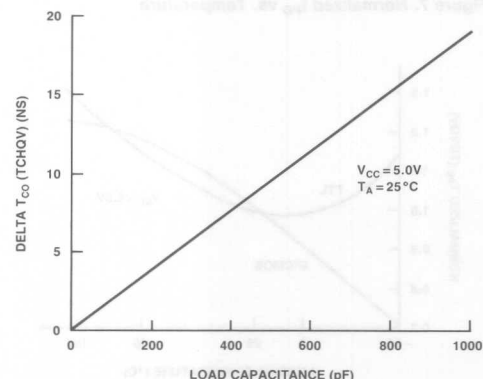


Figure 12. Delta  $t_{CO}$  vs. Output Loading



put drive characteristics ( $I_{OH}$  and  $I_{OL}$ ) are illustrated in Figures 13 and 14. The 200-ohm load line in Figure 13 clearly illustrates the ability of GAL devices to drive heavy loads and still maintain a solid  $V_{OH}$  level of approximately 2.9 volts.

### E<sup>2</sup>CMOS Input Characteristics

A schematic representation of the input translator/buffer utilized on all GAL devices is illustrated in Figure 15. The translator is specifically designed for compatibility with existing bipolar TTL circuit technologies, while providing significantly improved noise margins. Note the

presence of an internal decoupling capacitor ( $C_1$ ) physically integrated on each device input and I/O. Decoupling significantly improves transient-noise rejection ratio and makes it possible to use GAL devices successfully within noisy system environments, which would induce failures in other circuit technologies. Figure 16 is provided as an example of this 'designed-in' noise margin by illustrating a GAL16V8 operating as a counter with a 10MHz clock input signal that has only a 200mV swing. This is, of course, not recommended design practice, but demonstrates the sensitivity and noise-filtering capability of the GAL device input buffer.

Figure 13.  $I_{OH}$  vs. Output Voltage

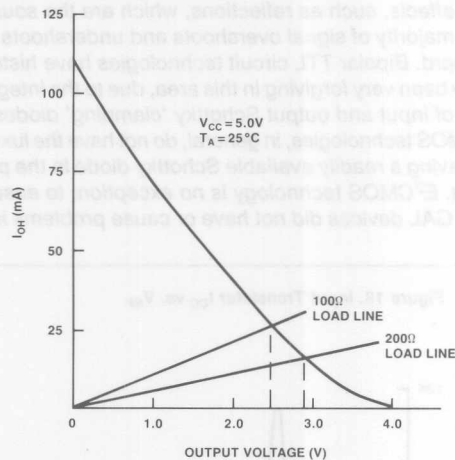


Figure 14.  $I_{OL}$  vs. Output Voltage

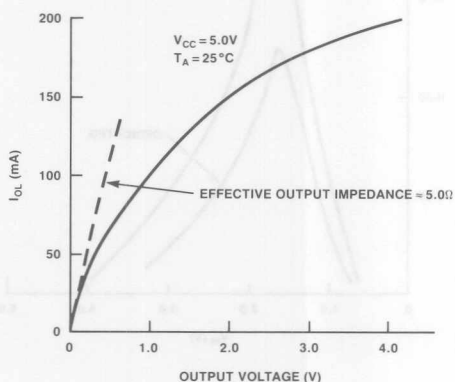


Figure 15. Input Translator/Buffer

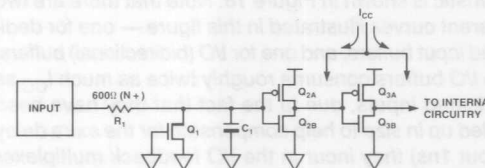
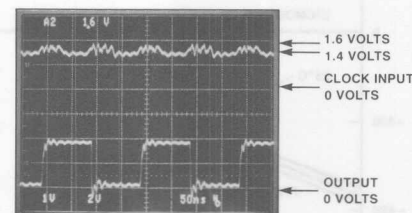


Figure 16. 10MHz Clocking with 200mV P-P Input

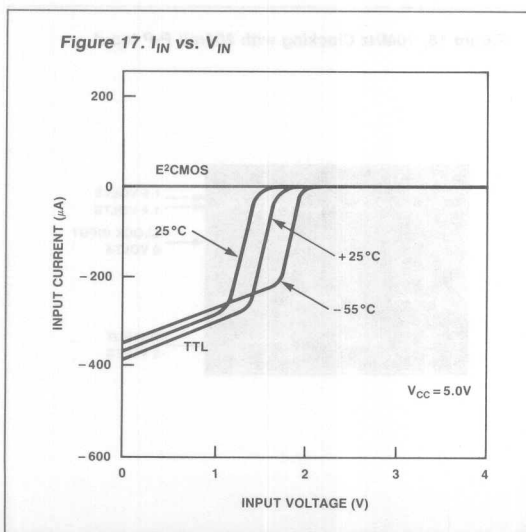


Two other important input characteristics of GAL devices are related to current-voltage (I-V) relationships. One of these is the input current ( $I_{IN}$ ) versus the input voltage ( $V_{IN}$ ), and is illustrated along with the common TTL characteristic in Figure 17. As can be readily discerned from the figure, E<sup>2</sup>CMOS provides the user with an ideal 'open circuit' input load characteristic, whereas the TTL devices present a significant DC load to the driving circuitry. Ten TTL devices in parallel can easily present a 4mA DC load to a driver.

The remaining I-V characteristic is related to the static  $I_{CC}$  consumed by the input translators as a function of the applied input voltage. As the input voltage transitions from a logic 0 to a logic 1, there exists a region typically between  $V_{IN} = 1$  to 3V, in which  $Q_{2A}$  and  $Q_{2B}$  are both slightly on and a current path exists between  $V_{CC}$  and  $V_{SS}$ . The same situation occurs with  $Q_{3A}$  and  $Q_{3B}$ , such that an E<sup>2</sup>CMOS input buffer characteristic will allow small amounts of DC current to flow as a function of input driving voltage levels. This input buffer characteristic is shown in Figure 18. Note that there are two different curves illustrated in this figure — one for dedicated input buffers, and one for I/O (bidirectional) buffers. The I/O buffers consume roughly twice as much  $I_{CC}$  as dedicated inputs, due to the fact that they have been scaled up in size to help compensate for the extra delay (about 1ns) they incur in the I/O feedback multiplexer circuit that dedicated inputs do not experience.

### ESD Protection Network

It should be noted that all GAL device inputs have an integral electrostatic discharge (ESD) protection structure comprising a diffused N+ resistor and a 'grounded-gate' transistor (refer to Figure 15) with a special phos-



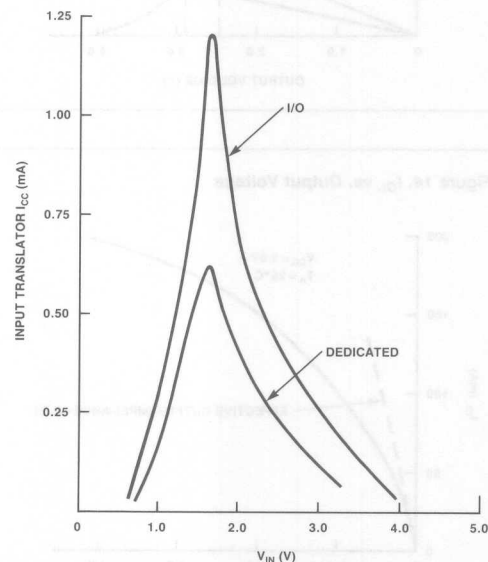
phorous implant in its drain region that is designed to non-destructively break down between source and drain at approximately 23 volts. This breakdown of  $Q_1$  provides a low-impedance path to ground, thus dissipating any potentially damaging charge that would otherwise pass to the internal circuitry. GAL device inputs are guaranteed to provide in excess of 1,500V of protection when tested according to MIL-STD-38510 ESD test methods.

### Overshoot Characteristics

With signal propagation delays decreasing as a result of device improvements, signal slew rates are approaching 5V/ns in some technologies. Proper attention to impedance-matching and termination becomes paramount, as system designers are faced with transmission-line effects, such as reflections, which are the source of a majority of signal overshoots and undershoots on a board. Bipolar TTL circuit technologies have historically been very forgiving in this area, due to the integration of input and output Schottky 'clamping' diodes.

MOS technologies, in general, do not have the luxury of having a readily available Schottky diode in the process. E<sup>2</sup>CMOS technology is no exception; to ensure that GAL devices did not have or cause problems in a

**Figure 18. Input Translator  $I_{CC}$  vs.  $V_{IN}$**



customer's system, Lattice design goals included a requirement to sustain large overshoots and undershoots on device pins while having no functional effect on the device performance. The reader is referred to Figure 19 which illustrates the same device and pattern as utilized in Figure 16, except the clock signal is now 16 volts peak-to-peak (-8.0V to +8.0V), still clocking the circuit at 10MHz with no device malfunction.

The specific I-V curves for input and output circuits are shown in Figures 20 and 21. The output circuit characteristic is very similar to that of TTL technologies. The input circuit can be seen to have a limited current-sourcing ability, as compared with TTL technology. □

Figure 19. 10MHz Clocking with 16V P-P Input

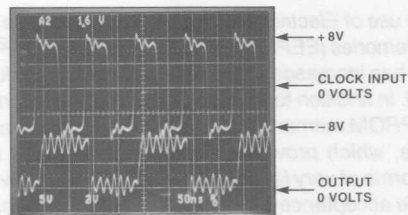


Figure 20. Output Voltage vs. Output Current

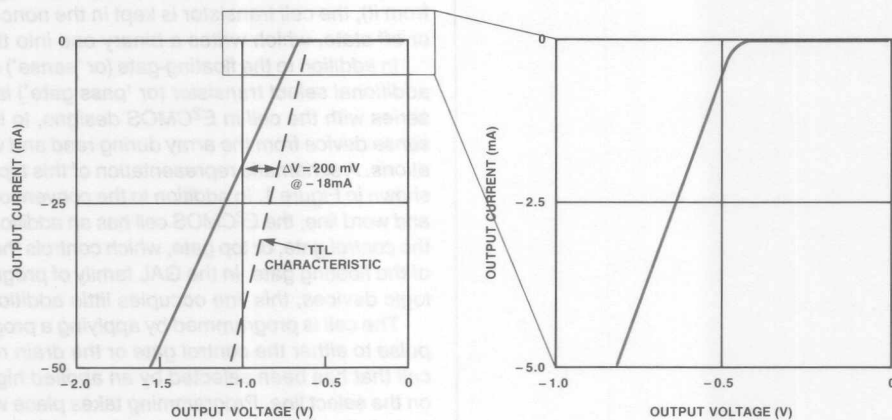
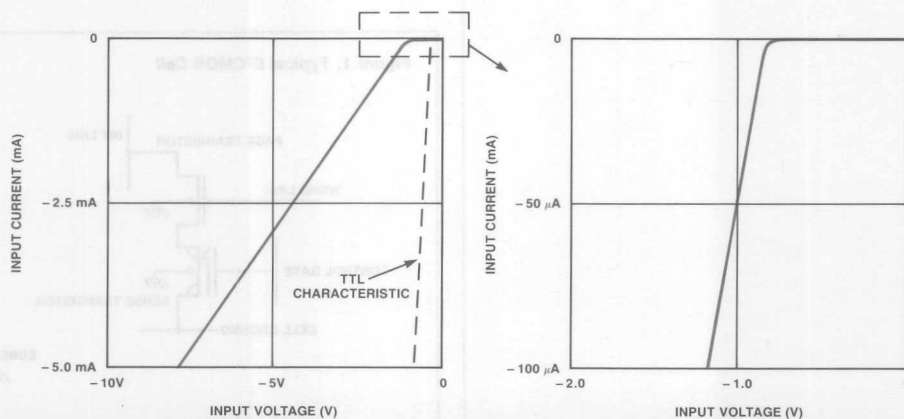


Figure 21. Input Voltage vs. Input Current





## INTRODUCTION

The use of Electrically Erasable Programmable Read Only Memories (EEPROMs) and the associated  $E^2$  technology has increased dramatically since its introduction in 1980. In addition to the standard use of the technology for EEPROM memories, the use of electrically erasable devices, which provide a flexibility unmatched by all other forms of very large-scale integration, is now gaining wide acceptance as the ultimate technology for configurable and programmable systems.

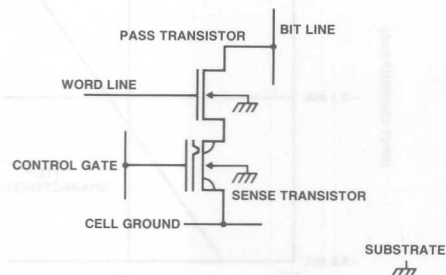
The widespread use of  $E^2$  technology has been hampered in the past by its need for ultra-clean processing. The technology requires on-chip high voltage, as well as fast switching devices — requirements that typically push the classical scaling laws of MOS devices. In addition to the restrictions on the technology, the use of electron tunneling through a high-quality oxide adds many new concerns about device reliability. In this report, the basic components of the  $E^2$  technology are reviewed, along with a detailed analysis of their operation. The results of an extensive modeling effort are presented, as well as a review of Lattice Semiconductor's use of the technology for reprogrammable logic families.

An  $E^2$  cell is built around an MOS transistor with a floating gate, which is externally charged or discharged by a small programming current. If the floating gate is charged up to a positive potential (by removing electrons from it), the cell transistor is kept in the nonconducting or off state, which writes a binary one into the cell.

In addition to the floating-gate (or 'sense') device, an additional select transistor (or 'pass gate') is added in series with the cell in  $E^2$ CMOS designs, to isolate the sense device from the array during read and write operations. A schematic representation of this type of cell is shown in Figure 1. In addition to the conventional bit line and word line, the  $E^2$ CMOS cell has an additional line for the control gate, or top gate, which controls the potential of the floating gate. In the GAL family of programmable logic devices, this line occupies little additional area.

The cell is programmed by applying a programming pulse to either the control gate or the drain region of a cell that has been selected by an applied high voltage on the select line. Programming takes place when electrons 'tunnel' through a thin dielectric, which is indicated in the schematic by the small notch in the floating gate over the drain of the sense device. Detailed operation of the  $E^2$ CMOS cell is described following a review of the requirements and tradeoffs of the process technology

Figure 1. Typical  $E^2$ CMOS Cell



## E<sup>2</sup>CMOS Technology

Lattice's E<sup>2</sup>CMOS technology is based on a combination of CMOS and NMOS technologies that successfully satisfies the requirements for both high-voltage and high-speed devices on chip. By combining 'pumped' (reverse-biased) substrate techniques and depletion devices from NMOS technology, with low-power CMOS devices, Lattice's E<sup>2</sup>CMOS technology maintains high performance while meeting the high-voltage requirements of programming.

In addition to combining the techniques of both NMOS and CMOS, E<sup>2</sup>CMOS technology incorporates an additional level of polysilicon to form the floating gate, as well as an ultra-clean, ultra-thin tunneling oxide of approximately 95 Angstroms' thickness. A microphotographed cross-section of the E<sup>2</sup>CMOS cell, as well as its schematic representation, is shown in Figure 2. Note that the sense-transistor channel length is defined by a masking layer and not by the polysilicon gate, as in the case of the select, or pass, transistor. This technique minimizes the size of the cell while maintaining a high coupling ratio between the floating gate and the control gate. As in the case of an ultraviolet-light erasable PROM (EPROM) cell, the E<sup>2</sup>CMOS cell requires an very low-leakage, high-quality oxide between the two levels of polysilicon to guarantee good data-retention characteristics.

A more dramatic illustration of the cell is shown in the scanning-electron-microscope (SEM) photograph of Figure 3, in which the various layers have been etched back to reveal the floating-gate structure.

The E<sup>2</sup>CMOS cell programs when a large voltage appears across the thin tunnel dielectric. The resulting tunneling current places electrons onto the floating gate, turning off the sense transistor. (With a reverse potential applied, electrons tunnel off the floating gate, turning on the sense transistor.) Once the charge has been placed

on the floating gate, the actual floating-gate potential can be modulated by the voltage on the control gate through capacitive coupling. It is this capacitive coupling that is used to generate the high voltage across the tunnel dielectric at the beginning of a programming pulse.

During a programming cycle, the cell is first erased to a 1, or nonconducting state, then selectively written to a 0, or conducting state, by a write cycle. This prevents the sense device from conducting current during the write operation, when voltage is applied to the drain of the device.

During an erase cycle, a large voltage is applied to the control gate of the cell to be programmed, as shown in Figure 4. If all current through the tunnel oxide is neglected, the floating gate simply tracks the applied voltage according to the relationship of a capacitive divider:

$$V_{fg} = C_{up} * V_{cg} + V_{fg}(0)$$

Figure 3. Etched Back E<sup>2</sup>CMOS Cell

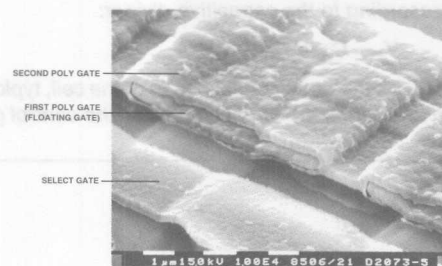
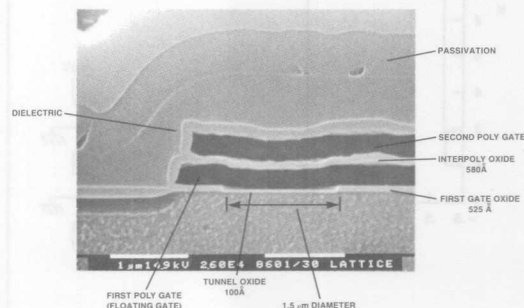
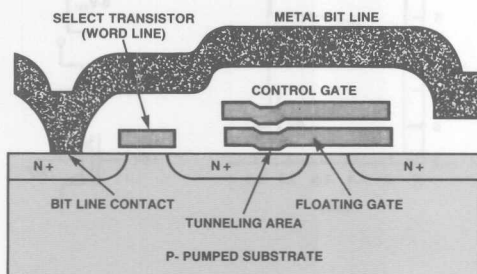


Figure 2. E<sup>2</sup>CMOS Cell Cross-Section



where  $C_{up}$  is the coupling ratio of the cell, typically between 0.6 and 0.7. At the end of the erase pulse, the floating gate again couples negatively by the same amount, ending up back at the initial floating-gate voltage  $V_{fg}(0)$ .

However, the large voltage applied across the tunnel dielectric discharges the floating gate during the erase pulse (Figure 4). At the end of the erase pulse, the floating gate ends up at a potential that is lower than the initial floating gate voltage by the amount that the floating gate has decayed during the pulse. This negative voltage is sufficient to turn off the sense transistor during a read operation. The magnitude of the control-gate voltage that is required to couple this negative floating-gate voltage up to the threshold of the sense device and actually turn it on after the erase pulse is defined as the programmed high threshold,  $V_{tHigh}$ .

### Write Cycle Programming

During the write cycle, a large voltage is applied to the bit line of the cell to be programmed, as shown in Figure 5. If current through the tunnel oxide is again neglected, the floating gate tracks the applied drain voltage according to the capacitive divider:

$$V_{fg} = C_d \cdot V_{drain} + V_{fg}(0)$$

where  $C_d$  is the drain coupling ratio of the cell, typically much lower than the coupling ratio to the control gate

and ranging between 0.2 and 0.3. As in the erase case, the floating gate again couples negatively by this same amount, and ends up back at the initial floating gate voltage,  $V_{fg}(0)$ , at the end of the write pulse. Also note that the pass transistor may have a voltage drop across it, lowering the voltage on the drain below the applied programming voltage,  $V_{pp}$ .

However, current indeed flows through the tunnel oxide, since the low coupling ratio keeps the floating gate at a low potential, forcing a large negative voltage across the tunnel oxide. The tunneling current charges the floating gate during the write pulse (Figure 5). At the end of the write pulse, the floating gate ends up at a potential that is higher than the initial floating gate voltage by the amount that the floating gate has charged during the pulse. This positive voltage is sufficient to turn on the sense transistor during a read operation. The magnitude of the control gate voltage that is required to couple this positive floating gate voltage down to the threshold of the sense device and actually turn it off is defined as the programmed low threshold,  $V_{tLow}$ .

### Reading the Cell

After an erase cycle, the charge on the floating gate has left the sense transistor in the off, or 1 state. If the erase cycle is followed by a write cycle, the floating gate charge leaves the sense device in the on, or 0 state. The data in the cell can be read, therefore, simply by sensing

Figure 4. Erase Pulse

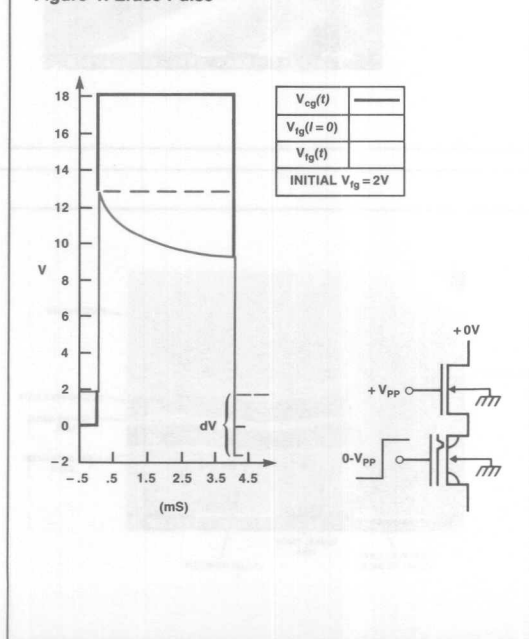
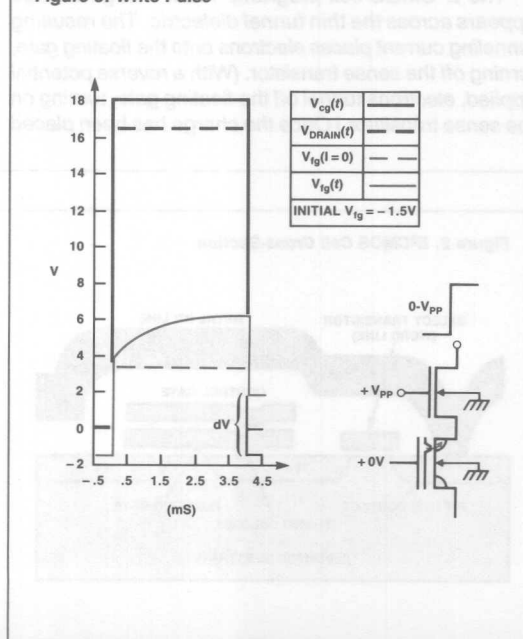


Figure 5. Write Pulse



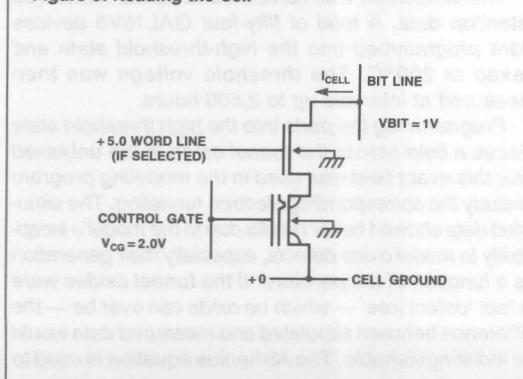
the cell current when biased with the control gate centered between the on and off states, as shown in Figure 6. The bit-line and control-gate voltages are selected to minimize the potential across the tunnel dielectric during a read in order to maximize the retention of the floating-gate charge. Using relatively low voltages, data can be retained for greater than 20 years of continuous reads. The actual magnitude of the programmed thresholds — and thus the margins of the cell — are controlled by the programming voltage, the physical cell layout, and the electrical characteristics of the tunnel oxide.

### Modeling the E<sup>2</sup>CMOS Cell

Accurate modeling of the E<sup>2</sup>CMOS cell requires an understanding of cell capacitance and coupling ratio, as well as the electrical properties of the tunnel oxide. Once these are understood, a model for the behavior of the cell under a variety of programming conditions is possible. Since coupling ratio and cell capacitance depend on the physical cell size and the oxide thicknesses grown during processing, for the model to be useful in studying the process sensitivity of the E<sup>2</sup>CMOS cell, it must be capable of simulating process variations. For this reason, we have developed a model that not only generates typical cell characteristics, but that takes into account all of the process sensitivities of the technology. This model can thus calculate both the mean and the variance, or standard deviation, of cell capacitance.

Once the statistical modeling of the physical cell is complete, it can be applied to the model of the electrical performance of the cell. Since we wish to evaluate the statistical properties of the cell, it is mandatory that our electrical model be fast — as well as accurate — for simulating the mean and standard deviation of programming characteristics. An analytical model has been developed that accurately predicts both the programmed threshold and the maximum field across the thin tunnel oxide.

Figure 6. Reading the Cell



The capacitive coupling ratio of the E<sup>2</sup>CMOS cell — with respect to both the control gate and to the bit line — can be calculated from a physical layout of the cell and the oxide thicknesses of all the overlapping areas. To calculate the coupling ratio for different process variations, shrink factors, and oxide thicknesses, the drawn cell is digitized and final end-of-process dimensions calculated on a computer. This permits studying the effect of process dimensions, misalignment, and thickness control on the resulting coupling ratios.

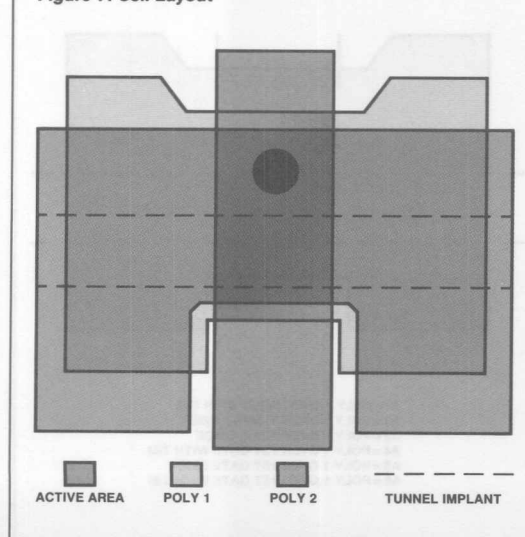
The basic 'as-drawn' cell of the GAL16V8 is shown in Figure 7. This cell is optimized for low-voltage programmability, wide programming margins, and guaranteed endurance of 100 cycles. The coupling ratio between the control gate and the floating gate,  $C_{cg}$ , can be calculated from the capacitive dividers formed by the poly 2 control gate, the poly 1 floating gate, and ground. The poly 1-to-poly 2 capacitance is simply:

$$C_{cg-fg} = C_1 = \frac{\epsilon_r \cdot \epsilon_o \cdot \text{Area}}{T_{oxIPO}}$$

where  $T_{oxIPO}$  is the thickness of the interpoly oxide,  $\epsilon_r = 3.9$  for oxide, and  $\epsilon_o$  is the dielectric constant. In contrast, the capacitance between the floating gate and ground is more complex, being made up of many small areas of different thicknesses, as shown in Figure 8. Taking all of the possible variations in oxide thickness into account we get the capacitance of the floating gate as:

$$C_{fg} = \sum_{i=1}^5 \frac{\epsilon_r \cdot \epsilon_o \cdot \text{Area}_i}{T_{ox_i}}$$

Figure 7. Cell Layout



Breaking these components up into three different types:

where

$C_2$  is the capacitance between the floating gate and the substrate ( $A_1 + A_3$ );

$C_3$  is the capacitance between the floating gate and the drain ( $A_2 + A_4 + 1/3 \cdot A_5$ );

$C_4$  is the capacitance between the floating gate and the source ( $A_6 + 2/3 \cdot A_5$ );

and  $C_1$  is the capacitance between the control gate and the floating gate. With these definitions, the coupling ratio between the control gate and floating gate becomes:

$$C_{up} = \frac{C_1}{C_1 + C_2 + C_3 + C_4}$$

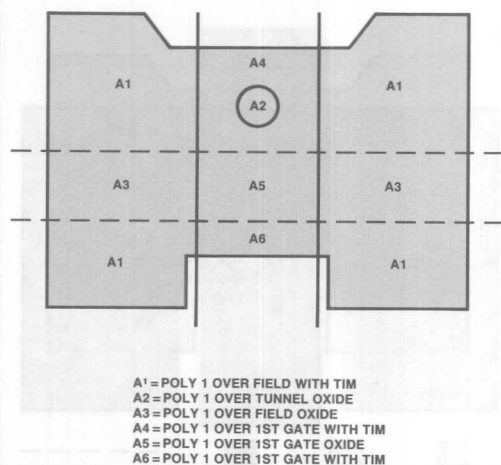
The coupling ratio between the drain and the floating gate can be calculated in a similar manner; however, the sense transistor will, in this case, source-follow, and automatically charge up the source capacitor  $C_4$  as the floating gate charges. Thus, the drain coupling ratio, from the  $n+$  diffusion to the floating gate, is simply:

$$C_d = \frac{C_3}{C_1 + C_2 + C_3}$$

### E<sup>2</sup> Cell Failure Mechanisms

The reliability of the E<sup>2</sup> technology is controlled by the characteristics of the tunnel dielectric and interpoly

Figure 8. Layout Showing Oxide Variations



oxide, as well as the additional stress placed on the gate oxide due to the on-chip high voltage. The high voltage requirements of the gate oxide impact the operating life of the technology and can be characterized by means of standard acceleration techniques. The reliability of the tunnel oxide, on the other hand, depends critically on the amount of charge that has passed through the dielectric, and thus degrades with cycling of the part. Characterization of this cycling-dependent reliability — or endurance — of the technology requires different methods.

### Data Retention Failures

The E<sup>2</sup>CMOS technology utilizes a floating gate in much the same way as an EPROM, and as such, is subject to many of the same failure modes. Excellent charge-retention characteristics have been proven with the EPROM technology, where the failure rate is very low. EPROM failures are typically single-bit failures, in which the cell can be programmed but cannot retain information over a long period of time. Such failures are due to charge loss or charge gain through defects in the interpoly oxide in the cell. Lattice's E<sup>2</sup>CMOS technology utilizes a high-temperature oxidation process that produces interpoly and tunnel oxides of extremely low defect density.

This type of data-retention failure can be effectively screened out, using the same standard manufacturing tests that have been developed for EPROMs such as high-temperature retention 'bakes' and cell threshold 'margining'.

The programmed-threshold margin test capability designed into the GAL16V8 and GAL20V8 simplifies the detection of threshold shifts caused by data-retention failure. To further examine threshold shifts due to charge loss or charge gain, a model was developed to simulate tunnel-oxide leakage, in which Fowler-Nordheim tunneling is the mechanism. The Fowler-Nordheim tunneling equation was used and its coefficients were extracted from dielectric 'tunnel-oxide' capacitors on test chips.

The simulation was correlated to actual measured retention data. A total of fifty-four GAL16V8 devices were programmed into the high-threshold state and baked at 200°C. The threshold voltage was then measured at intervals up to 2,500 hours.

Programming the parts into the high threshold state places a field across the tunnel oxide in the unbiased cell; this exact field was used in the modeling program to study the corresponding electron tunneling. The simulated data showed better results due to the model's incapability to model oxide defects, especially their generation as a function of temperature. If the tunnel oxides were in fact 'defect free' — which no oxide can ever be — the difference between simulated and measured data would be indistinguishable. The Arrhenius equation is used to



extrapolate from elevated-temperature data the time required for a corresponding threshold shift at room temperature. The results of the correlation are shown in Table 1.

### Peripheral Oxide Failures

The high voltage required to program the  $E^2$ CMOS cell — typically 16V to 20V — places an additional high field stress on the peripheral decoding and driver circuitry that controls the array during an erase or write cycle. The high coupling ratio of the GAL16V8 cell allows sufficient programming at 16 volts, in contrast with the 18V to 20V other  $E^2$  cells require. This reduces the high fields that gate oxides are exposed to and thus reduces the probability of breakdown. Gate-oxide breakdown is minimized in the GAL device family through careful control of the voltages used for programming the cell, and can be screened using standard acceleration techniques, such as dynamic burn-in.

### Read Disturb Failures

Although the  $E^2$ CMOS cell may be programmed at most hundreds or thousands of times, it must be capable of withstanding a near-infinite number of read cycles without suffering any disturbance of its programmed data. During the read operation, the low voltage applied between the control gate and the drain may induce very low Fowler-Nordheim leakage through the tunnel oxide that can alter the charge on the floating gate. Since the voltages during a read are carefully selected to minimize any potential read-disturb failures, it is standard practice to accelerate this failure rate by applying a voltage that is much higher than the values used under normal operation. The model for simulating data-retention was modified to simulate the read conditions used on the GAL16V8. This model utilizes the Fowler-Nordheim equation to determine the exact leakage that will occur — due strictly to tunneling — for the specified read conditions; the Fowler-Nordheim coefficients were extracted from tunnel-oxide I-V characteristics. Figure 9 shows the threshold shift (from an initial threshold of 7.5 Volts) that will occur over a million years due strictly to electron tunneling. As noted earlier, this model correlated with actual measured data, when used for modeling data retention. From the simulated and measured data, it is evident that GAL16V8 users will experience no problems associated

with charge loss — and hence, data retention — for the life of the part and much longer.

### Dynamic Cell Failures

The electrical characteristics of the thin tunnel dielectric will change with time as more and more charge has passed through it. Similarly, the characteristics of the defects that occur in the tunnel oxide will also change as an increasing amount of charge passes through the oxide. This results in a failure rate that is dependent on the number of erase and write programming cycles. There can be many types of cycle-dependent failures, from defect mechanisms to oxide wear-out, all of which contribute to degrading the endurance of the  $E^2$ CMOS cell.

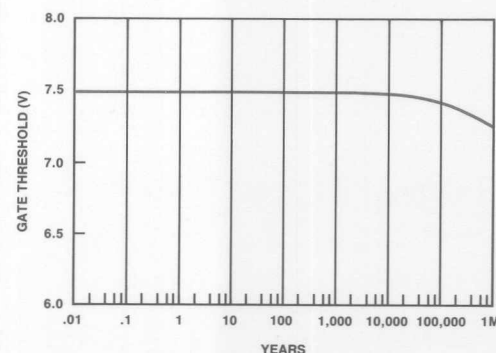
The simplest type of cycle-dependent failure is caused by a decrease in the programming efficiency as the number of cycles increases. Due to the degrading of the electrical characteristics of the tunnel oxide as more and more charge passes through it, the programming characteristics will also tend to degrade. As a result of this degradation, less current will flow through the oxide for a given field across it, and eventually the cell no longer sufficiently programs. This means that a higher programming voltage must be applied to obtain the same cell threshold. This degradation in the oxide, dubbed 'wear-out', which causes the programmed threshold to decrease as a function of the number of programming cycles is shown in Figure 10, where  $V_{th}^{High}$  and  $V_{th}^{Low}$  for a single  $E^2$ CMOS cell are plotted as a function of the number of cycles. This closing of the programmed window sets an upper bound on the endurance of the  $E^2$ CMOS

7

Table 1. GAL16V8 Threshold-Shift Data

Hours at 100°C	0	48	168	500	1,000	2,500
Measured Threshold	7.5	7.5	7.5	7.47	7.46	—
Simulated Threshold	7.5	7.5	7.5	7.4997	7.4994	7.4990
Years at 25°C	0	32	111	330	660	1,649

Figure 9. Threshold Shift at 25°C

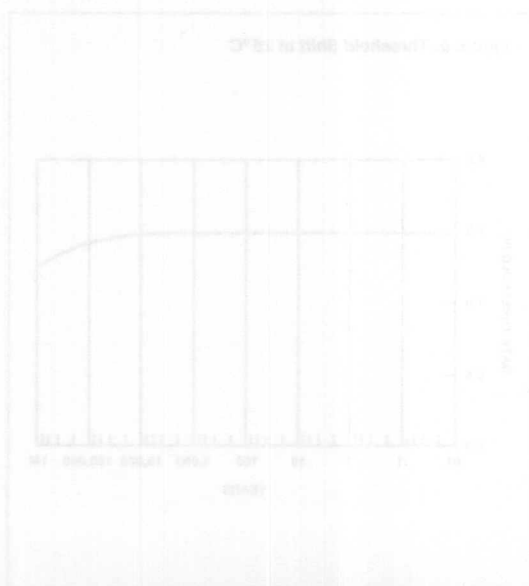
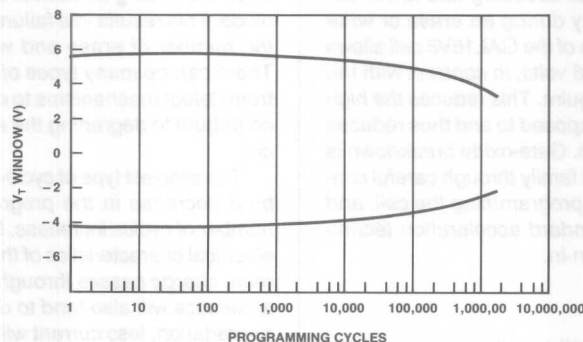




technology because of the loss of adequate margins on  $V_{THigh}$  and  $V_{TLow}$ . Note that the programming window does not begin closing significantly until after about

100,000 cycles. This poses no problem whatsoever for the GAL16V8 or GAL20V8, which are guaranteed for 100 cycles of programming. □

Figure 10. Threshold-Window Closing



The high voltage required to program the EEPROM is typically 18V to 20V — places an additional high field stress on the peripheral decoding and driver circuitry that controls the array. The high coupling ratio of the GAL16V8 and GAL20V8 cells requires 18V to 20V over 5V cells require. This means the high field stress on the peripheral decoding and driver circuitry that controls the array is increased to meet the requirements of the EEPROM. The high coupling ratio of the GAL16V8 and GAL20V8 cells requires 18V to 20V over 5V cells require. This means the high field stress on the peripheral decoding and driver circuitry that controls the array is increased to meet the requirements of the EEPROM.

Although the EEPROM cell may be programmed with most hundreds or thousands of times, it must be carefully monitored to avoid excessive wear. The high voltage required to program the EEPROM is typically 18V to 20V — places an additional high field stress on the peripheral decoding and driver circuitry that controls the array. The high coupling ratio of the GAL16V8 and GAL20V8 cells requires 18V to 20V over 5V cells require. This means the high field stress on the peripheral decoding and driver circuitry that controls the array is increased to meet the requirements of the EEPROM.

Table 1. GAL16V8 Threshold-Shift Data

Parameter	Units	Typical	Min	Max
Threshold Voltage	V	5.5	5.0	6.0
Threshold Voltage	V	5.5	5.0	6.0
Threshold Voltage	V	5.5	5.0	6.0
Threshold Voltage	V	5.5	5.0	6.0

## Lattice Quality Assurance

As part of the administration of the quality program, Lattice Quality Assurance:

- Assumes all manufacturing parameters that are critical to a product's quality and reliability are identified and remain in statistical process control; drives corrective action for those parameters not exhibiting statistical control.
- Executes the required auditing of both internal and external manufacturing operations and quality assurance programs; follows up on all corrective action, as appropriate.
- Assumes all critical operations involved in the manufacturing, testing or inspection of Lattice product are identified and executed only by trained operators; maintains all certification records of Lattice operators.
- Assumes all critical equipment involved in the manufacturing, testing or inspection of Lattice product is identified and subject to the requirements of the established calibration system.
- Assumes proper identification, review and disposal of product that is found to be nonconforming or has been quarantined; drives material review boards, as required.
- Establishes the procedures for the examination of failed devices identified in the qualification testing, inspection, customer returns or in process material; assumes each failure mechanism is addressed with appropriate corrective action.
- Maintains a document control system so as to provide the required controlled policies, procedures, specifications and product-specific instructions necessary to manufacture our products and maintain the Quality Assurance program; assumes proper version control and engineering change procedures.
- Reviews all proposed major changes in the manufacturing, testing or inspection of Lattice product, to ensure no changes are made without proper evaluation; assumes customer notification of product changes, as required.
- Establishes the required inspection steps and appropriate sample-plan implementation at each step, to ensure the necessary manufacturing quality control; assumes identified defective material is marked, corrected, and its disposition is appropriate.
- Assumes the accurate conversion of customer requirements into work instructions; issues Certificates of Conformance, as required, with product shipments.
- Specifies the appropriate electrical, mechanical and environmental qualification testing to evaluate mechanisms that may contribute to component failure.

It is the responsibility of Lattice Semiconductor's Quality Assurance to establish and maintain a quality assurance program with appropriate policies and procedures to meet our goals, as well as the goals of our customers. The program is commensurate with Lattice's Quality Assurance Manual. It is based on, and in compliance with, the product assurance program requirements of MIL-STD-883C, Appendix A, and the inspection and testing requirements specified in MIL-STD-883C.

Lattice enforces all such requirements on any and

INTRODUCTION	1
GAL DEVICE SPECIFICATIONS	2
LOGIC TUTORIAL	3
USING DEVELOPMENT TOOLS	4
GAL DEVICE APPLICATIONS	5
TECHNICAL BRIEFS	6
E <sup>2</sup> CMOS TECHNOLOGY OVERVIEW	7
GAL DEVICE QUALITY AND RELIABILITY	8
ARTICLE REPRINTS	9
APPENDICES	10
SALES OFFICES	11

## INTRODUCTION

It is the responsibility of Lattice Semiconductor's Quality Assurance to establish and maintain a quality-assurance program with appropriate policies and procedures to meet our goals, as well as the goals of our customers. The program is communicated via Lattice's Quality Assurance Manual. It is based on, and in compliance with, the product-assurance program requirements of MIL-M-38510, Appendix A, and the inspection-system requirements, per MIL-I-45208. This program assures compliance to the design, manufacture, inspection and testing requirements as specified in MIL-STD-883C.

Lattice enforces all such requirements on any and all subcontractors involved in the manufacture of its product, and is solely responsible for securing and proving the documentation and control of its Quality Assurance Program.

Lattice Quality Assurance is responsible for the continued review and update of its Quality Program to ensure that it continues to meet the standards as defined by the referenced specifications, customer requirements, and industry standards.

## Lattice Quality Assurance

As part of the administration of the quality program, Lattice Quality Assurance:

- Assures all manufacturing parameters that are critical to a product's quality and reliability are identified and remain in statistical process control; drives corrective action for those parameters not exhibiting statistical control.
- Executes the required auditing of both internal and external manufacturing operations and quality-assurance programs; follows up on all corrective action, as appropriate.
- Assures all critical operations involved in the manufacture, testing or inspection of Lattice product are identified and executed only by fully trained operators; maintains all certification records of Lattice operators.
- Assures all critical equipment involved in the manufacture, testing or inspection of Lattice product is identified and subject to the requirements of the established calibration system.
- Assures proper identification, review and dispositioning of product that is found to be nonconformant or has been quarantined; chairs material review boards, as required.
- Establishes the procedures for the examination of failed devices identified in the qualification testing, inspections, customer returns or in process material; assures each failure mechanism is addressed with appropriate corrective action.
- Maintains a document control system so as to provide the required controlled policies, procedures, specifications and product-specific instructions necessary to manufacture our products and maintain the Quality Assurance program; assures proper revision control and engineering change procedures.
- Reviews all proposed major changes in the manufacturing, testing or inspection of Lattice product, to assure no changes are made without proper evaluation; assures customer notification of product changes, as required.
- Establishes the required inspection steps and appropriate sample-plan implementation at each step, to assure the necessary manufacturing quality control; assures identified defective material is analyzed, corrective action is specified, and lot dispositioning is appropriate.
- Assures the accurate conversion of customer requirements into work instructions; issues Certificates of Conformance, as required, with product shipments.
- Specifies the appropriate electrical, mechanical and environmental qualification testing to evaluate mechanisms that may contribute to component failure rate. □

## The Lattice Quality Commitment

Since inception, Lattice Semiconductor Corp. has embraced the highest standards in all pursuits, from product-development objectives to the qualities and capabilities of its workforce. The goal of excellence in leading-edge CMOS products sets a corresponding demand for excellent product quality and reliability. Lattice's high-speed 256K CMOS static RAM, for example, integrates more than one million transistors — every one of which must operate for 20 years or longer without error and without failure.

Lattice attains its high standards of product quality and reliability in three basic ways. First, we believe strongly that quality is the responsibility of each and every individual involved with a product — from the very first stages of its development. As an example, all products begin with the design criterion of adequate margin to ensure 100% functionality over  $\pm 10\%$  power-supply voltage extremes and over the full-military temperature range of  $-55^{\circ}\text{C}$  to  $+125^{\circ}\text{C}$ . Indeed, it is our philosophy that quality and reliability are inherent in the design of every Lattice product.

Secondly, a Lattice product must pass an extensive qualification program before it is released to the marketplace. This includes compilation of accelerated life-test data at temperature and voltage extremes, as well as a battery of physical tests for package and chip environmental integrity. Moreover, once a Lattice product has been released to high-volume production, we continue to collect life-test data, to ensure that extrapolated failure rates and quality levels reflect actual in-system performance.

Thirdly, after a product is qualified, strict quality controls and monitors are applied to assure its quality level. Lattice's Quality Assurance program, which meets or exceeds all requirements outlined in MIL-M-38510, Appendix A, as well as all inspection system requirements in MIL-I-45208A, assures compliance to the design, manufacture, inspection, and testing requirements as specified in MIL-STD-883C. In the case where manufacturing processes are subcontracted, Lattice enforces all such requirements on the subcontractor through quality-control monitors and periodic audits, to ensure that they in fact adhere to specifications. Thus, customers can depend on the fact that Lattice is solely responsible for securing and proving the documentation and control of its Quality Assurance program.

We fully recognize that the degree of complexity exhibited in our products today mandates a veritable partnership between Lattice and our customers, to satisfy the formidable demands of quality and reliability. As we further drive the limits of VLSI to multimillion-transistor densities and submicron geometries, we encourage even more tightly coupled relationships, which we believe will be absolutely essential to transferring the benefits of our CMOS VLSI technologies to our customers.

## INTRODUCTION

Lattice Semiconductor maintains a comprehensive reliability-qualification program to assure that each product achieves its reliability goals. Data is continuously accumulated after initial qualification through monitor programs, so as to further drive failure rates down. Each product's qualification plan is generated in conformance to Lattice's Qualification Policy (Doc.#70-100164), with failure analysis in conformance to Lattice's Failure Analysis Procedures (Doc.#70-100166). Both documents are contained in Lattice's Quality Assurance Manual, which may be obtained upon request.

Failure rates in this qualification summary are expressed in FITS. Due to the very low failure rate of integrated circuits, it is convenient to refer to failures in a population during a period of  $10^9$  device hours; one failure in  $10^9$  device hours is defined as one FIT.

## High-Temperature Operating Lifetest

This test thermally accelerates those failure mechanisms that could occur as a result of operating the device continuously in an application. A pattern specifically designed to exercise the maximum amount of circuitry is programmed into the GAL device, and this pattern is continuously read while the supply voltage is maintained at 5.5V and the ambient temperature is held at 125°C.

A summary of the high-temperature operating lifetest results is shown in Table 1. One failure was observed in 297,500 device hours of 125°C operating life; this translates into an early-life failure rate of 283 FIT at 55°C, with a 60% upper-confidence level. This failure rate was calculated with a thermal activation energy of 0.5eV.

## High-Temperature Retention

This test specifically accelerates charge gain onto (or loss from) the floating gates in the GAL device's array. Since the charge on these gates determines the actual pattern and functions of the GAL device, this test is a measure of how reliably the device retains programmed information. It thus differs from the operating lifetest, which typically detects catastrophic functional failures. In high-temperature retention testing, the array of the device is programmed either to the high-threshold or low-threshold condition, and then baked at 150°C. The biasing is such that it represents the condition a device would see in a system application.

A summary of the high-temperature retention results is shown in Table 2. No failures were observed in 375,400 device hours of 150°C static-biased retention. This

Table 1. HTOL Reliability Data

HIGH-TEMPERATURE OPERATING LIFETEST GAL16V8 (125°C, 5.5V)				
Readout	48 Hrs.	168 Hrs.	500 Hrs.	1,000 Hrs.
Device Quantity	298	298	298	297
Failure Quantity	0	0	1	0

Table 2. Biased HTRB Reliability Data

HIGH-TEMPERATURE RETENTION GAL16V8 (150°C, 5.5V)				
Readout	48 Hrs.	168 Hrs.	500 Hrs.	1,000 Hrs.
High $V_T$				
Device Quantity	350	350	200	125
Failure Quantity	0	0	0	0
Low $V_T$				
Device Quantity	350	350	200	125
Failure Quantity	0	0	0	0



translates to an early-life failure rate of 20 FIT at 55°C with a 60% upper-confidence level. This failure rate was calculated with a thermal activation energy of 0.6eV.

A second retention evaluation was done to attempt to predict when a significant portion of the population would lose programmed data. For this evaluation, a smaller quantity of devices of various tunnel-oxide thicknesses (to represent the extremes of the manufacturing process) was used. Devices were programmed so as to detect charge loss and baked unbiased at 200°C for 2,500 hours.

A summary of these results is shown in Table 3. No failures were seen through 2,500 hours, which is equivalent to 193 years of life at 55°C (or 1,650 years at 25°C), assuming a 0.6eV activation energy. Projections of retention several orders of magnitude beyond this are possible, when one considers the actual charge loss on the floating gate through this evaluation. Actual cell thresholds (first failure and background failure) were moni-

tored at each readout. An average change of less than 100mV in cell threshold after 2,500 hours at 200°C was observed from the starting average of 7.17 volts. This agrees with theoretical charge loss projections through the tunnel oxide used in Lattice's GAL devices. Based on actual measurements of Fowler-Nordheim tunneling characteristics of our high-quality tunnel oxides, the charge loss of a floating gate with an initial threshold of 7.5V is shown in Figure 1. As can be seen, at 25°C, one would expect a MTTF due to charge loss that approaches 1 million years.

#### Moisture-Resistance Test

This test accelerates those mechanisms resulting from inadequate moisture barriers in the die and package construction. Pressure pot uses both pressure and temperature to accelerate penetration of moisture into the package and to the die surface. A summary of the pressure-pot data is shown in Table 4. No failures were observed through 168 hours at 121°C and 2 ATM pressure.

#### Temperature Cycling

This test accelerates failures resulting from mechanical stresses induced from differential thermal expansion of adjacent films, layers and metallurgical interfaces in the package and die. Devices are exposed to repeated cycling between -65°C and +150°C. Table 5 shows the results of temperature cycling on the GAL16V8. No failures were observed through 1,000 cycles of temperature cycling, which was performed in accordance with MIL-STD-883C, Method 1010, Condition C. □

Table 3. Unbiased HTRB Reliability Data

HIGH-TEMPERATURE RETENTION GAL16V8 (200°C, UNBIASED)					
Readout	48 Hrs.	168 Hrs.	500 Hrs.	1,000 Hrs.	2,500 Hrs.
Device Quantity	54	54	54	54	54
Failure Quantity	0	0	0	0	0

Figure 1. Threshold Shift at 25°C

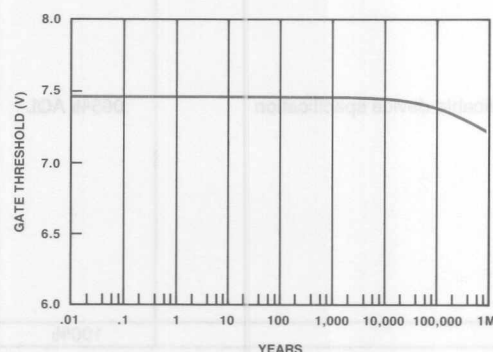


Table 4. Moisture-Resistance Reliability Data

PRESSURE POT GAL16V8 (121°C, 2ATM)		
Readout	48 Hrs.	168 Hrs.
Device Quantity	100	100
Failure Quantity	0	0

Table 5. Temperature-Cycling Reliability Data

TEMPERATURE CYCLING GAL16V8 (-65°C TO 150°C)				
Readout	10 Cyc	250 Cyc	500 Cyc	1,000 Cyc
Device Quantity	150	150	150	150
Failure Quantity	0	0	0	0



## DESCRIPTION

All HI-REL(X) grade products undergo demanding screening procedures according to the Methods and Requirements described in Table. 1. The HI-REL (X) processing includes 100% 160-hour burn-in at an

ambient temperature of +125°C, per Method 1015, followed by 100% temperature testing of all DC and AC parameters over the full military temperature range. All HI-REL (X) products are housed in hermetic ceramic packages, either DIP or leadless-chip-carrier (LCC) configurations. □

Table 1. HI-REL (X) Screening Flow

Screen	Test Method	Requirement
<b>INCOMING INSPECTION</b>		
Wafer Lot Acceptance	Per LATTICE Spec.	3 wafers/lot
<b>VISUAL AND MECHANICAL</b>		
Internal visual	2010, Condition B	100%
Stabilization bake	1008, Condition C 24 hours	100%
Temperature cycle	1010, Condition C	100%
Constant acceleration	2001	100%
Visual inspection	5004	100%
Hemeticity, Fine and Gross	1014	100%
<b>BURN-IN</b>		
Interim (pre burn-in) electrical	Per applicable device specification at 25°C	100%
Burn-in	1015, 160 hours at 125°C or equivalent	100%
Post-burn-in electrical	Per applicable device specification at 25°C	100%
Percent Defective Allowable	5004	5%
<b>FINAL ELECTRICAL TESTS</b>		
Static (dc)	At temperature and power supply extremes (T <sub>A</sub> = -55°C to +125°C) (V <sub>CC</sub> = 4.5V to 5.5V)	100%
Functional		
Switching (ac) or Dynamic		
<b>QUALITY CONFORMANCE INSPECTION</b> (Group A)		
A2: Static test at maximum rated operating temperature	Per applicable device specification	.065% AQL
A5: Dynamic Tests at maximum rated operating temperature		
A8: Functional tests at maximum and minimum rated operating temperatures		
A10: Switching tests at maximum rated operating temperatures		
A3: Static tests at maximum rated operating temperature		
A6: Dynamic tests at maximum rated operating temperature		
A11: Switching tests at maximum rated operating temperature		
<b>EXTERNAL VISUAL</b>	2009	100%

### DESCRIPTION

By specifying Lattice fully compliant MIL-STD-883C components, the user can be assured of receiving products that are manufactured, tested, and inspected according to the highest standards, for those applications where quality and reliability are vital. For special customer specifications or quality requirements — such as SEM analysis, X-ray, or other screening flows to meet specific needs — please contact your local sales office or Lattice Semiconductor directly at 1-800-FASTGAL.

All Lattice Semiconductor components processed in full compliance to MIL-STD-883C are first screened according to the procedures defined in Method 5004 Class B. This includes 100% 160-hour burn-in at an ambient temperature of + 125°C per Method 1015, followed by 100% temperature testing of all DC and AC parameters over the full – 55°C to + 125°C temperature range, as described in Table 1. Samples of the product that have been screened are then submitted to the Quality Conformance procedures as defined in Method 5005 Class B. These Quality Conformance inspections include Group A (Electrical), Group B (Mechanical), Group C (Chip Integrity), and Group D (Package Environmental Integrity), as described in Table 2. □

**Table 1. MIL-STD-883C Screening Flow**

Screen	Method	Requirement
Wafer Lot Acceptance	Per LATTICE Spec.	3 Wafers/Lot
SEM	Per LATTICE Spec.	1 Wafer/Lot
Internal Visual	2010 Cond. B	100%
Stabilization Bake	1008 24 HR Cond. C	100%
Temp. Cycling	1010 Cond. C	100%
Constant Acceleration	2001 Cond. E	100%
Visual Inspection	5004	100%
Hermeticity Fine Gross	1014 Cond. A1 Cond. C	100%
Pre Burn-In Electrical	Applicable Device Spec. TA = 25°C	100%
Burn-In	1015, 160 HRS. 125°C	100%
Post Burn-In Electrical	Applicable Device Spec. TA = 25°C	100%
Percent Defective Allowable	5004	5%
Final Electrical Test	Applicable Device Spec. TA = 125°C	100%
Final Electrical Test	Applicable Device Spec. TA = – 55°C	100%
QCI Sample Selection	MIL-M-38510F Section 4.5	Sample
External Visual	2009	100%

**Table 2. MIL-STD-883C Quality Conformance Testing**

Group	Method	Sample
<b>A: ELECTRICAL TESTS</b>		
GROUP A1, A4, A7, A9 Electrical Test	Applicable Device Spec. 25°C	.065 AQL
GROUP A2, A5, A8, A10 Electrical Test	Applicable Device Spec. Max. Operating Temp.	.065 AQL
GROUP A3, A6, A11 Electrical Test	Applicable Device Spec. Min. Operating Temp.	.065 AQL
<b>B: MECHANICAL TESTS</b>		
GROUP B1 Physical Dimension	2016	2(0)
GROUP B2 Resistance To Solvents	2015	4(0)
GROUP B3 Solderability	2003	LTPD = 15
GROUP B4 Internal Vision and Mechanical	2014	LTPD = 5
GROUP B5 Bond Strength	2011	LTPD = 5
GROUP B7 Hermeticity	1014	LTPD = 15
<b>C: CHIP INTEGRITY TESTS</b>		
GROUP C1 Dynamic Life Test	1005, 1,000 HRS. 125°C	LTPD = 15
End Point Electrical	Applicable Device Spec.	
GROUP C2 Temp Cycling	1010, Cond. C	LTPD = 15
Constant Acceleration	2001, Cond. E	
Hermeticity	1014	
Visual Examination	1010	
End Point Electrical	Applicable Device Spec.	
<b>D: PACKAGE ENVIRONMENTAL INTEGRITY</b>		
GROUP D1 Physical Dimensions	2016	LTPD = 15
GROUP D2 Lead Integrity	2004, Cond. B	LTPD = 15
Hermeticity	1014	
GROUP D3 Thermal Shock	1011, Cond. B, 15 Cycles	LTPD = 15
Temp. Cycle	1010, Cond. C, 100 Cycles	
Moisture Resistance	1004	
Hermeticity	1014	
Visual Examination	1004, 1010	
Endpoint Electrical	Applicable Device Spec.	
GROUP D4 Mechanical Shock	2002, Cond. B	LTPD = 15
Vibration	2007, Cond. A	
Constant Acceleration	2001, Cond. E	
Hermeticity	1014	
Visual Examination	1004, 1010	
Endpoint Electrical	Applicable Device Spec.	
GROUP D5 Salt Atmosphere	1009, Cond. A	LTPD = 15
Hermeticity	1014	
Visual Examination	1009	
GROUP D6 Internal Water Vapor	1018 < 5,000 PPM at 100°C	3(0)
GROUP D7 Adhesion to Lead Finish	2025	LTPD = 15



INTRODUCTION	1
GAL DEVICE SPECIFICATIONS	2
LOGIC TUTORIAL	3
USING DEVELOPMENT TOOLS	4
GAL DEVICE APPLICATIONS	5
TECHNICAL BRIEFS	6
E <sup>2</sup> CMOS TECHNOLOGY OVERVIEW	7
GAL DEVICE QUALITY AND RELIABILITY	8
ARTICLE REPRINTS	9
APPENDICES	10
SALES OFFICES	11



# EEPROM TECHNOLOGY SEEDS REPROGRAMMABLE LOGIC

LATTICE'S GAL16V5 USES A HIGH-SPEED CMOS PROCESS

TO REALIZE A REPROGRAMMABLE LOGIC ARRAY

Technology used in Lattice's reprogrammable logic arrays is not only powerful, when the growing popularity of reprogrammable logic arrays is considered, it is also a new thing in the logic array as well. Lattice's GAL16V5 uses a high-speed CMOS process to build reprogrammable logic devices. The technology used in Lattice's EEPROM technology is also a new thing in the logic array as well. Lattice's EEPROM technology is also a new thing in the logic array as well.

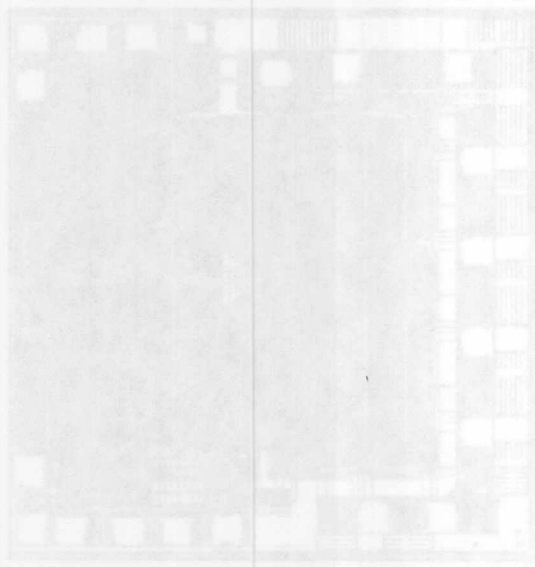
Technology used in Lattice's reprogrammable logic arrays is not only powerful, when the growing popularity of reprogrammable logic arrays is considered, it is also a new thing in the logic array as well. Lattice's GAL16V5 uses a high-speed CMOS process to build reprogrammable logic devices. The technology used in Lattice's EEPROM technology is also a new thing in the logic array as well. Lattice's EEPROM technology is also a new thing in the logic array as well.

## LATTICE

## EEPROM TECHNOLOGY SEEDS PROGRAMMABLE LOGIC

Although it is a new thing in the logic array as well, Lattice's EEPROM technology is not only powerful, when the growing popularity of reprogrammable logic arrays is considered, it is also a new thing in the logic array as well. Lattice's EEPROM technology is also a new thing in the logic array as well. Lattice's EEPROM technology is also a new thing in the logic array as well.

The implementation of a reprogrammable logic device in the logic array as well, Lattice's EEPROM technology is not only powerful, when the growing popularity of reprogrammable logic arrays is considered, it is also a new thing in the logic array as well. Lattice's EEPROM technology is also a new thing in the logic array as well.



Technology used in Lattice's reprogrammable logic arrays is not only powerful, when the growing popularity of reprogrammable logic arrays is considered, it is also a new thing in the logic array as well. Lattice's GAL16V5 uses a high-speed CMOS process to build reprogrammable logic devices. The technology used in Lattice's EEPROM technology is also a new thing in the logic array as well. Lattice's EEPROM technology is also a new thing in the logic array as well.

This article is reprinted from **Electronics Week June 3, 1985. Copyright © 1985 McGraw-Hill, Inc.** □

Technology used in Lattice's reprogrammable logic arrays is not only powerful, when the growing popularity of reprogrammable logic arrays is considered, it is also a new thing in the logic array as well. Lattice's GAL16V5 uses a high-speed CMOS process to build reprogrammable logic devices. The technology used in Lattice's EEPROM technology is also a new thing in the logic array as well. Lattice's EEPROM technology is also a new thing in the logic array as well.

Technology used in Lattice's reprogrammable logic arrays is not only powerful, when the growing popularity of reprogrammable logic arrays is considered, it is also a new thing in the logic array as well. Lattice's GAL16V5 uses a high-speed CMOS process to build reprogrammable logic devices. The technology used in Lattice's EEPROM technology is also a new thing in the logic array as well. Lattice's EEPROM technology is also a new thing in the logic array as well.

Technology used in Lattice's reprogrammable logic arrays is not only powerful, when the growing popularity of reprogrammable logic arrays is considered, it is also a new thing in the logic array as well. Lattice's GAL16V5 uses a high-speed CMOS process to build reprogrammable logic devices. The technology used in Lattice's EEPROM technology is also a new thing in the logic array as well. Lattice's EEPROM technology is also a new thing in the logic array as well.

Technology used in Lattice's reprogrammable logic arrays is not only powerful, when the growing popularity of reprogrammable logic arrays is considered, it is also a new thing in the logic array as well. Lattice's GAL16V5 uses a high-speed CMOS process to build reprogrammable logic devices. The technology used in Lattice's EEPROM technology is also a new thing in the logic array as well. Lattice's EEPROM technology is also a new thing in the logic array as well.

Technology used in Lattice's reprogrammable logic arrays is not only powerful, when the growing popularity of reprogrammable logic arrays is considered, it is also a new thing in the logic array as well. Lattice's GAL16V5 uses a high-speed CMOS process to build reprogrammable logic devices. The technology used in Lattice's EEPROM technology is also a new thing in the logic array as well. Lattice's EEPROM technology is also a new thing in the logic array as well.

Technology used in Lattice's reprogrammable logic arrays is not only powerful, when the growing popularity of reprogrammable logic arrays is considered, it is also a new thing in the logic array as well. Lattice's GAL16V5 uses a high-speed CMOS process to build reprogrammable logic devices. The technology used in Lattice's EEPROM technology is also a new thing in the logic array as well. Lattice's EEPROM technology is also a new thing in the logic array as well.



# EEPROM TECHNOLOGY SEEDS REPROGRAMMABLE LOGIC

## LATTICE'S GAL16V8 USES A HIGH-SPEED CMOS PROCESS TO REALIZE A REPROGRAMMABLE LOGIC ARRAY

**T**he technology used in electrically erasable programmable read-only memories, which are growing in popularity wherever nonvolatility and erasability are needed, is now hitting pay dirt in the logic arena as well. Designers at Lattice Semiconductor Corp., Portland, Ore., have paired EEPROM technology with a high-speed CMOS process to build programmable logic devices. The programmable chip relies on EEPROM cells for an electrically alterable architecture so it can directly replace any of 21 device architectures in the 20-pin bipolar PAL family manufactured by Monolithic Memories Inc.

The GAL16V8 (for generic array logic) is said to run at bipolar speeds—25-ns propagation delay—with an output drive of 24 mA, using EEPROM cells in place of normal fuses (see photo below). This results in an electrically reprogrammable, reusable device that can be programmed on standard logic-array programmers using supporting software such as ABEL, which is provided by Data I/O Corp. in Redmond, Wash., and CUPL, provided by Assisted Technology Inc. in San Jose, Calif.

GAL16V8s can be used repeatedly during system prototyping. The same GAL16V8 used in one type of logic circuit can be reprogrammed for use in another circuit. Because of its output logic architecture, each individual output can be tailored as registered, combinational, active high, active low, input only, or bidirectional.

Dean Suhr, manager of Lattice's Programmable Logic Product Engineering Division, says that the EEPROM cell in the GAL16V8 exhibits excellent data retention and endurance characteristics. He also says that the cell's design has been proven at Lattice in several large memory chips, including a 45-ns 64-K EEPROM.

"Unlike bipolar fusible-link devices, the 16V8 cells require no openings in the passivation for fuse debris to escape, nor are they subject to the 'grow-back' phenomenon of fuses," he notes. "This results in improved reliability." In addition, the cell writes quickly: an erase-and-program cycle of a 16V8 cell takes 20 ms.

Programmable logic devices place a stringent requirement on technology: circuits need to be extremely fast, current drive extremely high, and programming elements extremely reliable. Fusible-link bipolar technology once stood alone in meeting those requirements.

But Rahul Sud, chief executive officer of Lattice, says that is no longer the case. "Today, high-speed electrically erasable CMOS is the technology of choice for programmable logic," he told *ElectronicsWeek*.

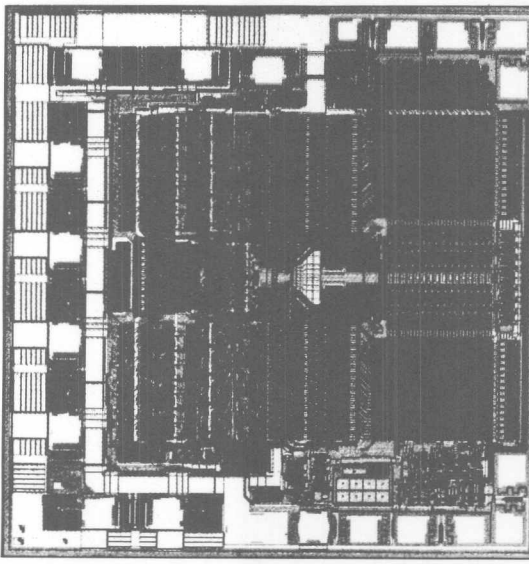
Sud calls his process UltraMOS. He points out the advantages of EEPROM cells for the nonvolatile programming elements over fuse links: a nondestructive nature for reusability and, thus, testability; the low programming current required,

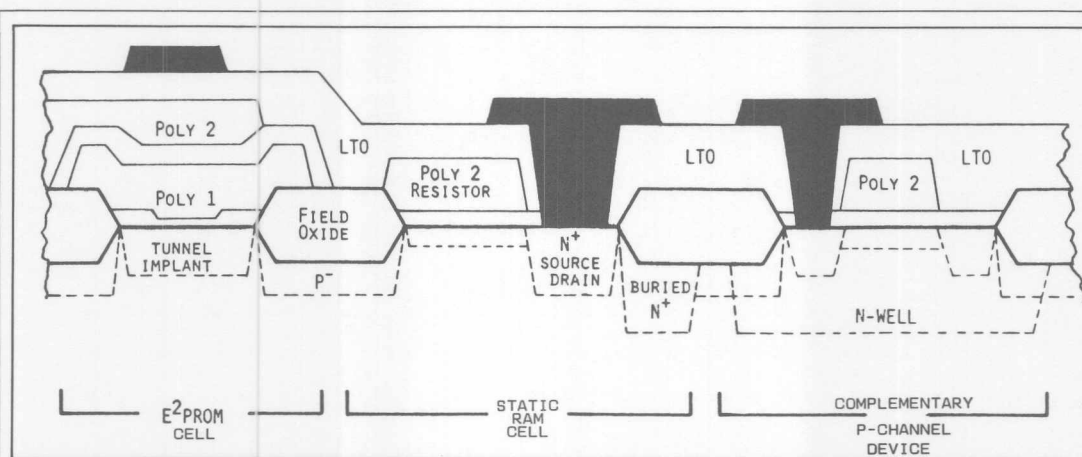
making possible the on-chip programming-signal generation key to next-generation in-circuit-alterable components; and reduced contamination through complete passivation of the chip, promoting greater reliability.

Some CMOS-chip makers have chosen EPROM cells, which require ultraviolet-light exposure for erasing, as the programming element in their PLAs. Although it has certain advantages over bipolar fuses, EPROM technology falls short of EEPROM when it is applied to programmable logic (see table); it is also not applicable to future architectures that will require in-circuit reprogrammability.

Sud explains that from the manufacturer's standpoint, EEPROM technology holds significant cost advantages over EPROM technology in building programmable logic devices. Although he admits it is a more complex process, EEPROM technology "affords significant savings in the cost of testing, since cells can be programmed, erased, and reprogrammed in milliseconds, whereas EPROM cells require 20-min ultraviolet-light exposures for erasure." Moreover, he continues, "though EEPROM is a less dense technology than EPROM, memory-array area has little impact on die size in a PLA, where some 90% of the chip comprises random logic."

The implications of a reconfigurable logic device in the engineering lab environment are significant. In the past, every design revision, checkout, or upgrade required pulling a





**Modular.** UltraMOS has been designed modularly, so steps in the flow may be eliminated if elements such as the load resistors are not needed. The process uses the first level for floating-gate EEPROM cells and the second level for load resistors and enhancement and depletion gates.

new PLA from an inventory stock of some 21 different architectures, with three speed/power options for each architecture. The PLA was patterned with the new logic, then tested for the first time for full functionality.

The old PLA was discarded because its fuse-link cells could not be reused. The GAL16V8 eliminates the need for the expensive inventory that is required for full-line support, thanks to its single-device replacement approach, 100% fuse-map compatibility, and performance that is superior to the current bipolar data devices.

The 16V8 can be repatterned and reused in less than a second, as opposed to the 15 to 20 minutes that are usually required. The EEPROM cells can be patterned for worst-case data-sheet performance during testing to fully guarantee, through actual test, performance over all data-sheet conditions without having to resort to correlated or simulated paths in the device.

"Because the GAL 16V8 is compatible with the 21 most popular programmable logic arrays, it can be substituted into existing system designs in a completely transparent fashion," according to Suhr. "The replacement is 100%, in terms of performance specifications, logic functionality, and fuse-map compatibility."

These same inventory and universal configurability benefits are available to the production and field-service staff. They, too, can employ the GAL16V8 as a single, reusable replacement for dozens of PLAs. Field modifications, service, upgrades, and functional changes can be made with only one device type.

The GAL16V8's output logic macrocell (OLMC) allows it to be configured in a number of ways that are not possible with ordinary PLAs, according to Lattice engineers. For example, because a synchronous (registered) output can be placed on any or all output pins, the user can create a variety of devices—a 16R1 or a 16R5, for example. These architectures are not possible with the ordinary PLAs that are currently available, say Lattice designers.

Lattice's UltraMOS is a basic CMOS EEPROM process that adds modules for

high-resistivity polysilicon loads and for a buried n+ interconnect layer. A double-polysilicon process, UltraMOS (see figure) uses the first level for floating-gate EEPROM cells and the second level for all regular enhancement and depletion gates, as well as for high-value load resistors.

UltraMOS has been designed in a modular fashion. Therefore, steps in its flow may be eliminated with no impact to temperature cycles if certain elements, such as the polysilicon load resistors, are not desired.

Minimum features of the process extend down to 1.25  $\mu\text{m}$ , with the smallest gate length currently limited to 1.5  $\mu\text{m}$ . As PLAs move to 24-, 40-, and 84-pin densities, the package and device reliability, power dissipation, and exceedingly long programming time for the larger arrays all serve to dissuade further pursuit of this technology.

User-programmable logic continues to enjoy steady growth: sales this year are expected to approach \$400 million. According to Sud, the reason for this growth is mainly time to market—computer-aided-design tools cannot compete with off-the-shelf silicon for turnaround time. The market for programmable logic is expected to hit more than \$1.5 billion by the end of the decade, with the lion's share going to Monolithic Memories. The GAL16V8 targets the millions of 20-pin PAL sockets. □

COMPARING LATTICE'S GAL AND MONOLITHIC MEMORIES' PAL

	GAL 16V8	PAL devices
Speed	25/35 ns	25/35 ns
Maximum active current	90 mA	180/90 mA
90 mA at 25-ns specification	Yes	No
Technology	CMOS, E <sup>2</sup> cells	Bipolar, fuse link
Reprogrammable	Yes	No
Fully tested fuse array	Yes	No
Guaranteed reliability	100%	99%
Devices needed to supply full line	1	More than 20
User signature	64 bits	None
Configurable architecture	Yes	No
"Odd" architectures available	Yes	No
Configurable outputs	Yes	No
Security	Visual and electronic	Electronic
Programming time	3 s	12 s



# LATTICE

## A 16ns CMOS EEPLA WITH REPROGRAMMABLE ARCHITECTURE

**David L. Rutledge  
John E. Turner  
Roy D. Darling  
Gregg R. Josephson**

This paper was presented at the 1986 IEEE International Solid-State Circuits Conference at the California Pavilion, February 21, 1986, and appeared in the IEEE Digest of Technical Papers. Copyright © 1986 IEEE. □

9

## SESSION XVIII: LOGIC ARRAYS AND MEMORIES

### FAM 18.1: A 16ns CMOS EEPLA with Reprogrammable Architecture

David L. Rutledge, John E. Turner, Roy D. Darling, Gregg R. Josephson

Lattice Semiconductor Corp.

Beaverton, OR

OVER THE PAST 10 YEARS, a number of programmable logic architectures and devices have been developed and reported, the majority of which have been implemented in bipolar technology<sup>1,2</sup>. This technology has the advantage of high-functional speed at the expense of relatively high-power consumption and lack of flexibility (one-time programmability). Architectural flexibility (I/O ratio, registered/asynchronous logic, output polarity) was only available through hardwired mask options until recent developments in bipolar technologies have produced one-time field configurable architectures<sup>3</sup>. MOS floating gate technologies (EPROM/EEPROM) have been utilized to address the reprogrammability and reconfigurability issues, but these devices have typically suffered tremendous speed impact (100ns) to achieve the flexibility<sup>4,5,6</sup>.

This paper will report on the development of a CMOS EEPLA device implemented in a floating gate process which achieves bipolar performance levels at reduced power levels. The EE memory technology has also been utilized to realize a device that is not only reprogrammable, but also architecturally reconfigurable.

Functionally, the device consists of a user-programmable *AND* array with up to 16 logical inputs. The *AND* array generates 64 product terms (pt) which are distributed 8 pt/output through a hardwired eight-input *OR* gate in each output logic macrocell (OLMC). A programmable architecture word consisting of 8b provides the configuration control data. Figure 1 illustrates the OLMC and how it utilizes the architecture word to alter the device architecture.

The goal of developing a reprogrammable, high-performance, low-power circuit mandated the choice of an EEPROM based CMOS technology. Manufacturing cost objectives dictated a conservative single metal, double-poly process. The technology development objectives are overviewed in Table 1. A key focus of the process development was to allow for high performance, low-voltage devices (5V) to be integrated with lower performance, high-voltage devices (18V). The goal was attained with the addition of a single *Junction-Grading* implant step. Depletion devices were included to accommodate the high speed sense amplifier and simplify high voltage circuit design.

The traditional bipolar *fuse* has been replaced by a nonvolatile reprogrammable EEPROM floating-gate type structure. The basic cell consists of two transistors, a select gate (input line), and a non-

volatile sense transistor. Each cell measures 0.44 square mils. The cell is repeated 2048 times to make up the entire programmable *AND* matrix organized as 32 input lines and 64 product terms.

Programming is performed in a minimum amount of time by parallel programming 64 cells (an entire row) with a single 10ms program pulse. Data to be programmed into each cell is shifted in via a Serial Register Latch (SRL). Following programming, verification can be performed by interrogating the state of each cell using the normal sense amplifier. Verification circuitry allows the sensed data to be loaded into the SRL which is then serially shifted out for validation. The direction of data flow (program or verify) is controlled by the PGM/VERIFY pass gates connected to the SRL as shown in Figure 2.

Common to all programmable logic arrays is the possibility of multiple input lines switching simultaneously. The switching of input lines is, in turn, coupled to the product term by Miller Capacitance. By placing the EE memory element between the sense amplifier and the select gate (Figure 3), input line switching noise is prevented from being coupled to the product term. As a result, a quieter sense level is achieved, inhibiting disturbance of the high speed sense amplifier.

Sensing of the product term is accomplished by a high-speed, single-ended current sensing technique. Sense currents as low as 5 $\mu$ A are obtainable at reasonable speed. The voltage swing of the product term is limited to approximately 100mV through the use of self-biasing negative feedback.

On-chip substrate bias generation ( $\sim 2.5V$ ) is utilized to provide: (1) latch-up immunity without EPI; (2) reduced sub-threshold field transistor leakage, and (3) reduced body-effect on high performance device thresholds.

Testability is a major consideration with any programmable logic device. This circuit employs four techniques which allow for virtually 100% testability of the device:

(1) EE memory technology allows for 100% testability of every cell in the array. This is in contrast to one-time programmable technologies such as bipolar, plastic One Time Programmable EPROM PLA (OOTPEPLA) which must use test rows and columns for *correlation*.

(2) A Logic Test Mode allows one to load data into the SRL serially for all 82 columns, and force these data onto the sense amp inputs (override the matrix data). This permits a very fast test of chip logic without requiring the array to be programmed. Many different *Apparent Patterns* can be rapidly tested in this way.

(3) Register preload is accomplished through the use of the SDIN and SDOUT buffers. Assertion of Preload (PRLD) (a supervoltage test signal) transforms the OLMC registers into shift registers and serial data are clocked in using the normal clock signal. This feature allows for complete controllability and observability of machine states.

(4) Margin Test Mode is provided to allow the internal memory control gate reference (MCG) to be overridden and driven externally to allow cell margin tests to be performed.

Speed and active power dissipation are illustrated in Figure 4. Typical device waveforms are presented in Figure 5.

The authors wish to thank A. Nguyen, K. Campbell and D. Tennant for graphics support, and J. Olund, B. Cremen, R. Van Art, B. Delepine, E. DeMuizon and P. Spadini for wafer processing.

<sup>1</sup> Monolithic Memories, Inc., *Bipolar LSI Databook*, Chapter 7; 1984.

<sup>2</sup> Signetics Corp., *Integrated Fuse Logic Data Manual*; 1983.

<sup>3</sup> Advanced Micro Devices, *Programmable Array Logic Handbook*; 1983.

<sup>4</sup> Wood, R.A., Hsieh, Y., Price, C.A., Wang, P.P., "An Electrically Alterable PLA for Fast Turn-Around-Time VLSI Development Hardware", *IEEE JSSC*, Vol. SC-16, No. 5; Oct., 1981.

<sup>5</sup> Leung, R., Lee, S.M., "A 50ns 48 Term Erasable Programmable Logic Array", *ISSCC DIGEST OF TECHNICAL PAPERS*, p. 130-131; Feb., 1985.

<sup>6</sup> Cuppens, R., Hartgring, D., Verwey, J., Peek, H., "An EEPROM for Microprocessors and Custom Logic", *ISSCC DIGEST OF TECHNICAL PAPERS*, p. 268-269; Feb., 1984.



See Page 8-10 for Fig. 2  
8-11 for Fig. 6

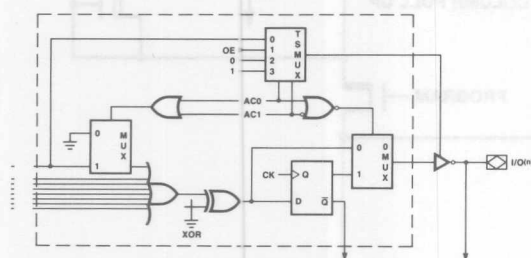


FIGURE 1—Output Logic Macrocell (OLMC).

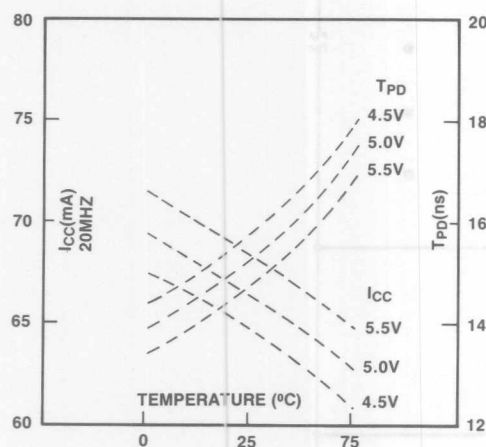


FIGURE 4—Speed and active power dissipation.

#### PROCESS

N-Well, CMOS  
Double-Level Poly-Si  
Single-Level Si-Al

#### CHANNEL LENGTH

NMOS 1.2 $\mu$ m  
PMOS 1.3 $\mu$ m  
High Voltage 3.0 $\mu$ m

#### OXIDE THICKNESS

Tunnel Oxide 100 Å  
First Gate 525 Å  
Interpoly 580 Å  
Second Gate 400 Å

POLYSILICON PITCH 4.5 $\mu$ m

METAL PITCH 5.4 $\mu$ m

CONTACT CUT  $\cdot 1.5\mu\text{m} \times 1.5\mu\text{m}$

TABLE 1—Process overview.

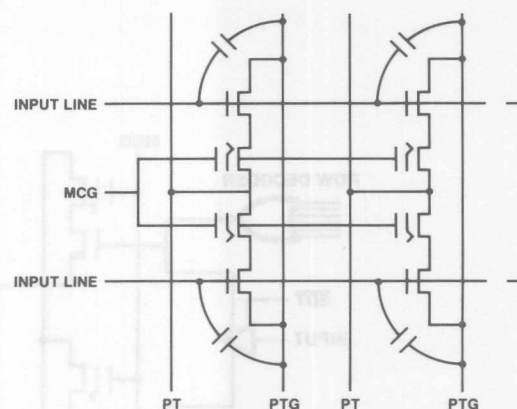
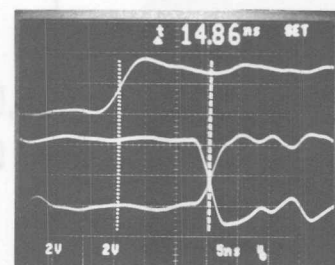
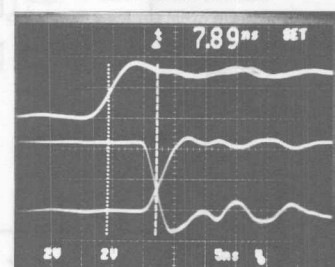


FIGURE 3—Schematic of row coupling to product term in conventional EEPROM cell. EEPLA isolates this coupling capacitance with EE element between sense amplifier and select gate.



INPUT TO OUTPUT DELAY



CLOCK TO OUTPUT DELAY

FIGURE 5—Typical device waveforms.

INPUT TO OUTPUT DELAY	16ns
CLOCK TO OUTPUT DELAY	8.5ns
STANDBY CURRENT	40mA
ACTIVE CURRENT (20MHz)	75mA
$I_{OL}/I_{OH}$	24mA/V - 3.2mA
INPUT/OUTPUT	TTL COMPATIBLE
DIE SIZE	116 x 122 mil <sup>2</sup>
PACKAGE SIZE	300-mil 20-pin DIP
	20-pin Square LCC/PLCC
RETENTION	> 10 Years
MAXIMUM W/E CYCLES	100
CELL SIZE	0.44 mil <sup>2</sup> /bit
PROGRAMMING TIME/CELL	10ms
PROGRAMMING TIME/ENTIRE ARRAY	< 400ms
PROGRAMMING VOLTAGE	16.5V

TABLE 2—Device characteristics.



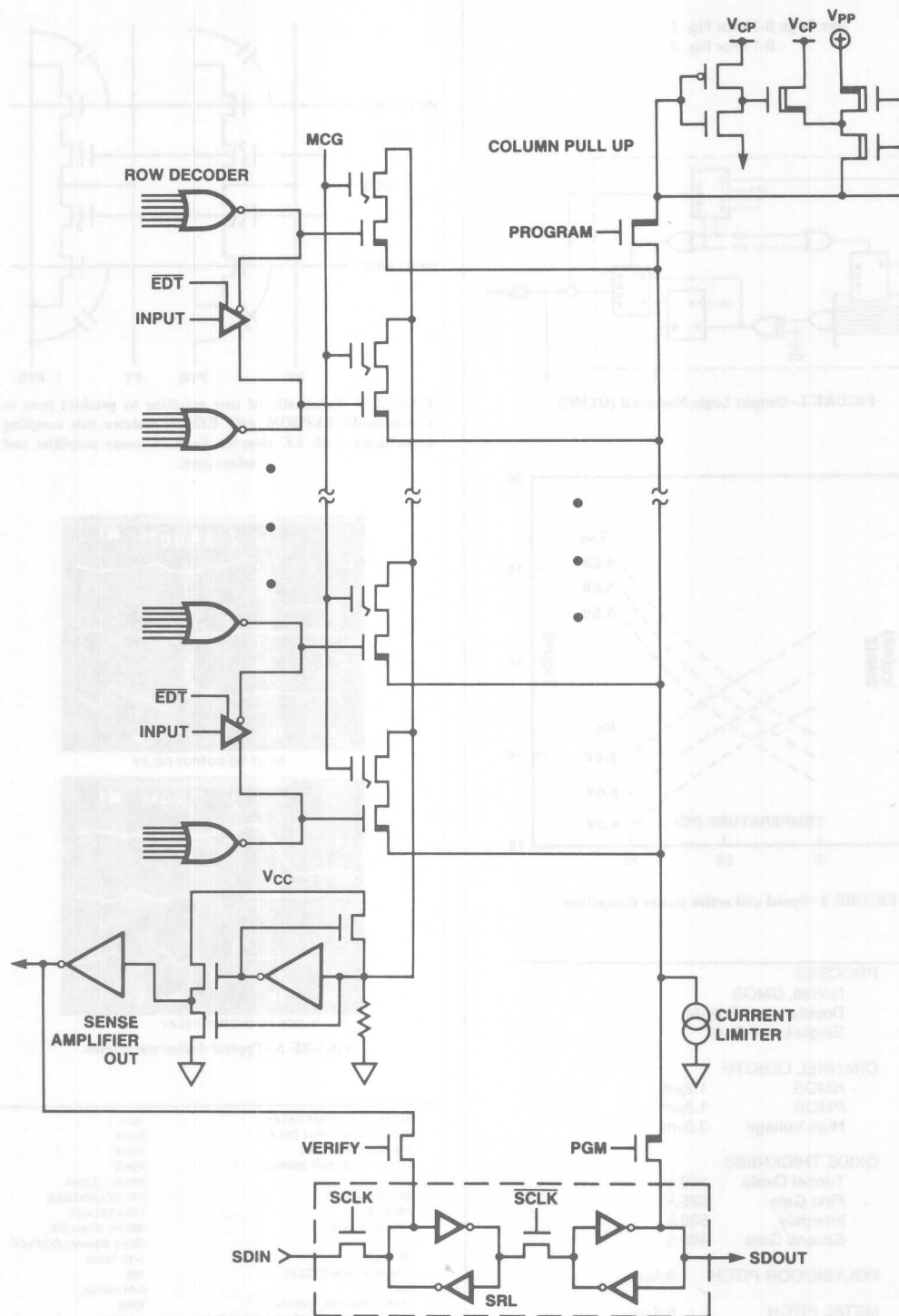
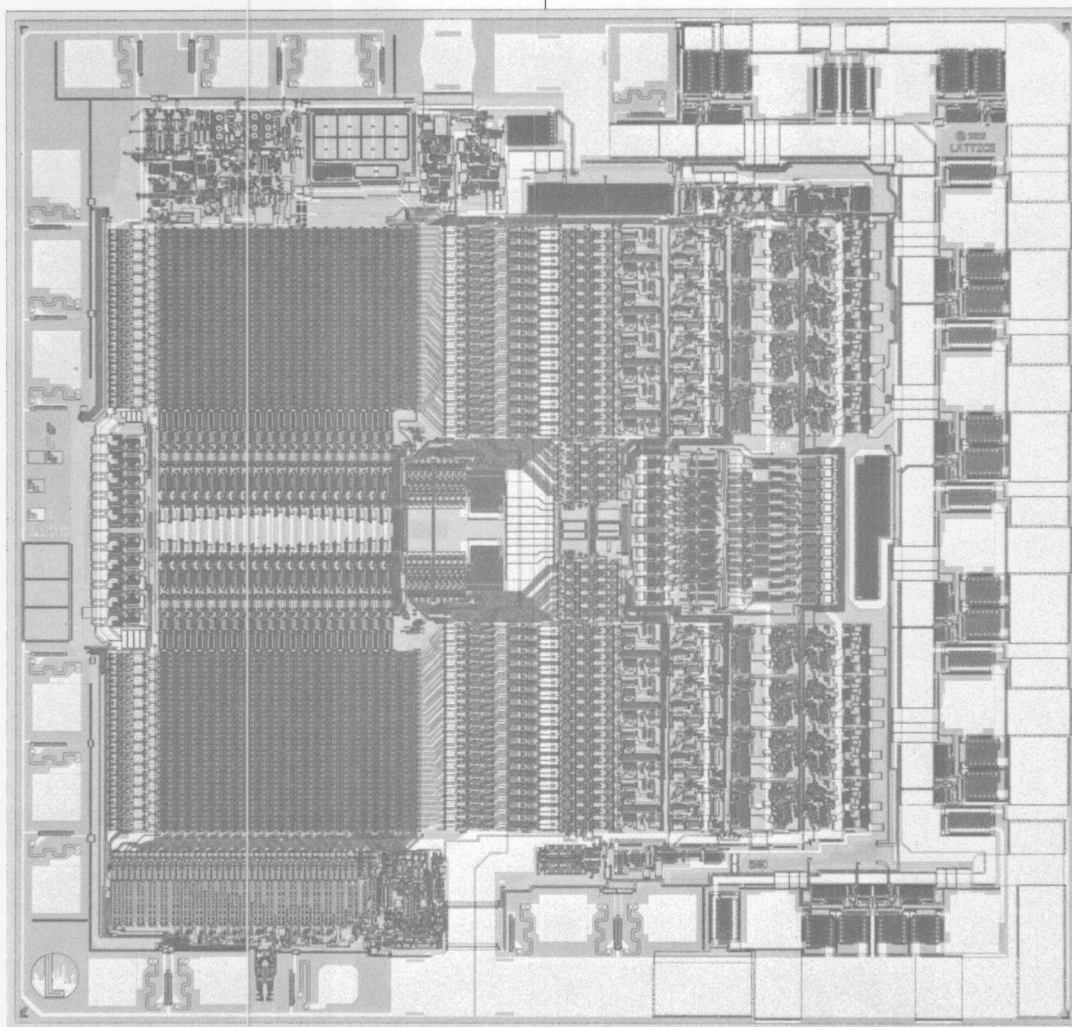
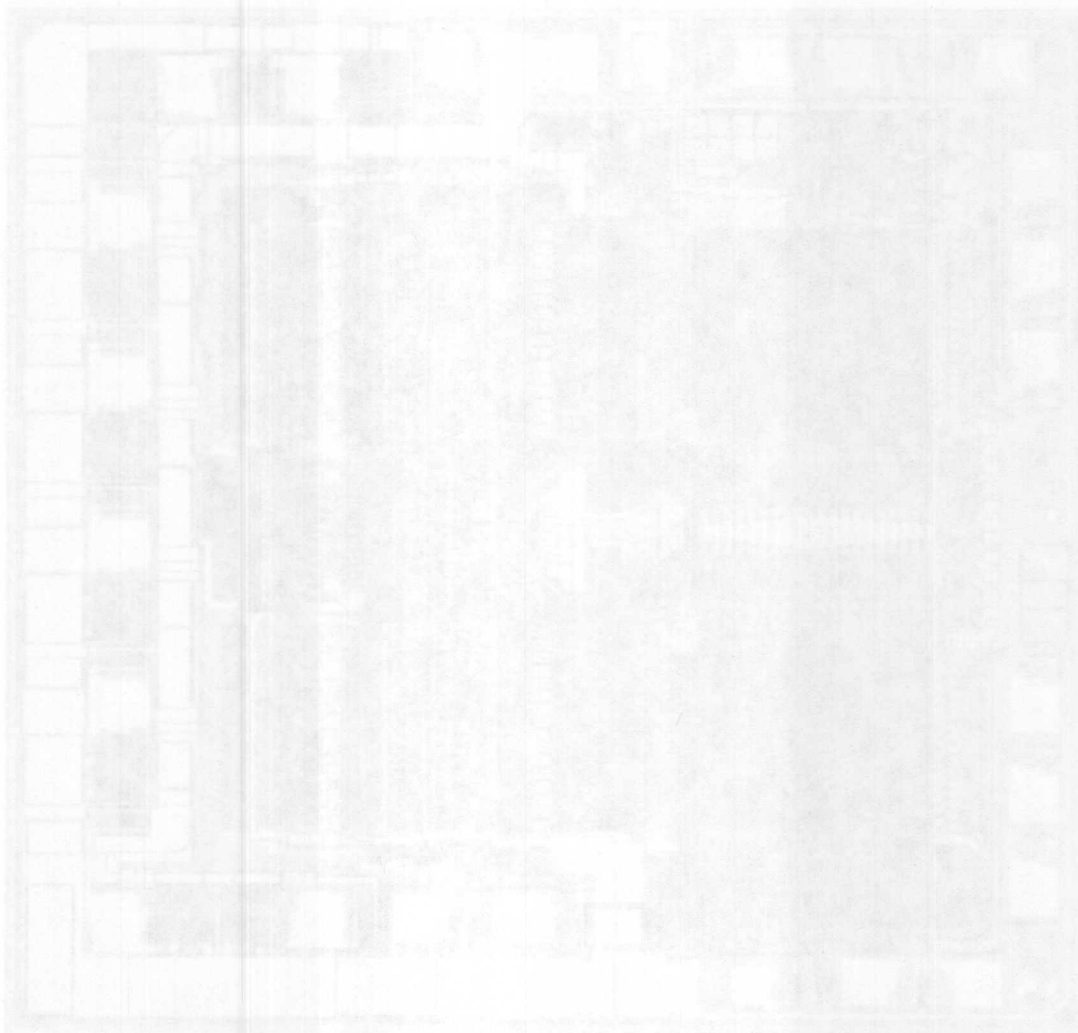


FIGURE 2—Diagram showing pass gates to control direction of data flow to Serial Register Latch (SRL).





# LATTICE

## ELECTRICALLY ERASABLE PLDs PROVIDE FULL TESTABILITY

Dean Suhr

This paper was presented at Northcon '85 Electronics Show & Convention in Portland, Oregon, and appeared in the Northcon/85 Professional Program Session Record, Session, 12. Copyright © Electronics Conventions Management, Inc.

and translated in into a prototype to verify the proper operation of all of the various components in the system.

What responsibility should the system designer have to validate that the PLD itself can program and perform in the database applications? The answer to this question traditionally has not been quite so clear.

With modern devices the manufacturer has assumed the burden for device performance validation and provided a 100% database functional device. Testing PLD's proved to be quite a challenge, however, due to the programmable nature of the device and the fact that the array is loaded into the PLD without ready to external pins that control gates. The inherent connection of the device from uncommitted logic to a custom pattern had the manufacturer in a quandary. The uncommitted logic typically was in a "do-nothing" configuration prior to the array being patterned. This left the manufacturer with having to be satisfied or guaranteed the device performance. The implication of this approach is that PLD's are "too difficult to test or have) no allowance for testing."

The responsibility for testing the PLD falls in various modes depending on whether you are a manufacturer or a user of the device. Each company generally expects the other to have fully tested the device and to assume responsibility for the successful handling of the failure.

The manufacturer generally contends that the device has been tested "completely" and that some failure (typically 1-2%) is expected during the user's programmed and test operation. The manufacturers have traditionally (by default) placed the burden of guaranteeing the database performance of the device on the user. Of course, the user would like to have the device completely tested and have guaranteed performance and 100% change when processing the device.

The manufacturers generally contend that the device has been tested "completely" and that some failure (typically 1-2%) is expected during the user's programmed and test operation. The manufacturers have traditionally (by default) placed the burden of guaranteeing the database performance of the device on the user. Of course, the user would like to have the device completely tested and have guaranteed performance and 100% change when processing the device.

ELECTRICALLY ERASABLE DEVICES PROVIDE FULL MANUFACTURER AND USER  
TESTABILITY FOR OPTIMAL PROGRAMMABLE LOGIC DEVICE QUALITY

Dean Suhr  
Product Marketing Manager  
Programmable Logic Products  
LATTICE SEMICONDUCTOR CORP.  
15400 NW Greenbrier Pkwy  
Beaverton, OR 97006  
(503) 629-2131

INTRODUCTION

Many aspects and "solutions" to the test issues of programmable logic devices (PLD) have been discussed over the past year or two. There are two basic reasons to test your programmable logic devices; first, to validate that logic equations (or any other input format) used to pattern the array performs the desired logical function, and second, to validate that the PLD is performing as the array pattern and datasheet specify.

The responsibility for testing the PLD falls in various courts depending on whether you are a manufacturer or a user of the devices. Each camp generally expects the other to have fully tested the devices and to assume responsibility for the associated handling of the fallout.

Who's Responsible for What

The manufacturers generally contend that the devices have been tested "completely" and that some fallout (typically 1-3%) is expected during the user's programming and test operations. The manufacturers have traditionally (by default) placed the burden of guaranteeing the datasheet performance of the devices on the user. Of course, the users would like to have the devices completely tested and have guaranteed performance and 100% throughput when processing the devices.

It is reasonably clear that the systems designer is responsible for the validation of his logic function (design verification) assuming a good integrated circuit (IC). He is also responsible for taking this simulation and transforming it into a prototype to verify the proper interaction of all of the various components in his system.

What responsibility should the systems designer have to validate that the PLD itself can program and performs to the datasheet specifications? The answer to this question traditionally has not been quite as clear.

With discrete devices the manufacturer has assumed the burden for device performance validation and provided a 100% datasheet functional device. Testing PLD's proved to be quite a challenge, however, due to the programmable nature of the device and the fact that the array is buried inside the die without ready access to external pins thru normal paths. The inherent conversion of the device from uncommitted logic to a custom pattern had the manufacturers in a quandary. The uncommitted logic typically was in a "do-nothing" configuration prior to the array being patterned. This left the manufacturer with having to correlate or guarantee the device performance. The implication of this approach is that PLD's are "too difficult to test or (have) no allowance for testing"<sup>1</sup>.

PROM's

The first programmable logic device, the PROM, was not as sophisticated as current programmable logic devices, and there were still real concerns with testing the array for programmability and final AC switching performance prior to patterning the array. The PROM array was not embedded. It was directly accessible to the external pins thru normal paths. As a result of this direct access PROM's were 100% testable after programming so there was always a point where the device could be verified against the datasheet specifications. This responsibility fell again onto the user.

Programmable Logic

PLD's, on the other hand, use the array to drive additional internal logic circuitry. The access to the array for programming uses special circuitry that is not in the device's data path during normal operating conditions. Recent attempts to more fully test this logic circuitry have resulted in more advanced test schemes. None of the schemes developed for the standard Bipolar fuse-link technology allow the manufacturer to actually test the programmability of all fuses. Nor do these schemes allow for test of the actual switching paths under normal signal stimulation conditions.

THE CHALLENGE

The challenge before us is to provide a programmable logic device that enhances the ability of the designer to implement his custom logic pattern without having to worry about the final device meeting the datasheet specifications. The device should be preverified for programmability and should have all functional and AC paths tested for switching performance. Correlations and guarantees are for the marketing and purchasing personnel. Engineering requires 100% tested devices.

THE RESPONSE

Lattice Semiconductor Corporation introduced the first **Electrically Erasable (E<sup>2</sup>) Programmable Logic Device**, the **GAL<sup>tm</sup>-16V8**, in March of 1985. This device is a 20 pin PLD that is fully compatible with industry standard PAL<sup>2</sup> devices. The generic approach taken in the design of the GAL-16V8 resulted in what we call the **Generic Array Logic<sup>tm</sup> (GAL<sup>tm</sup>)** family of devices which consists of both 20 and 24 pin devices in both generic and fixed architecture configurations.

The GAL family of devices are designed to exceed the performance of their Bipolar and E-PROM based counterparts. These devices offer a fully tested 25 nS max. propagation delay at 470 mW max. power dissipation (90 mA).

In addition, the GAL-16V8 and GAL-20V8 can be architecturally configured to be a logical equivalents to more than 20 PAL type architectures in their respective 20 and 24 pin configurations. They can also be configured to random architectures such as a 16R1 or a 18P2) by utilizing the output logic macrocell on each of the output pins to individually select the polarity, registers and feedback of each pin.

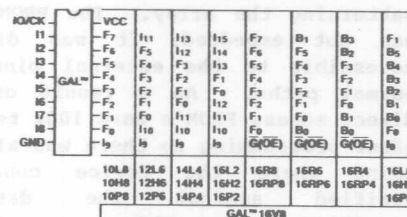
The device can be patterned using standard JEDEC fuse maps such as those produced by PALASM<sup>3</sup> or CUPL<sup>4</sup> or ABEL<sup>5</sup>. These software packages will provide all the information for complete patterning of the GAL devices, allowing the user to exploit the configuration flexibility of the device.

The array pattern can also be duplicated from a pre-programmed master device supplied by any PAL device manufacturer. The algorithm implemented on approved programming hardware such as the Data I/O LogicPak<sup>5</sup>, Stag PPZ<sup>6</sup> or Valley Data Sciences Model 160 generic programmers resolves all pattern & configuration issues at the hardware level without having to deal with any higher level software assembler or compiler.



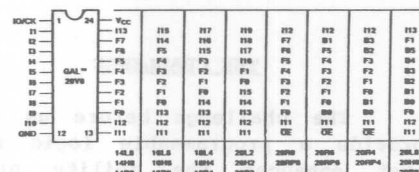
## LATTICE 20 PIN PRODUCTS

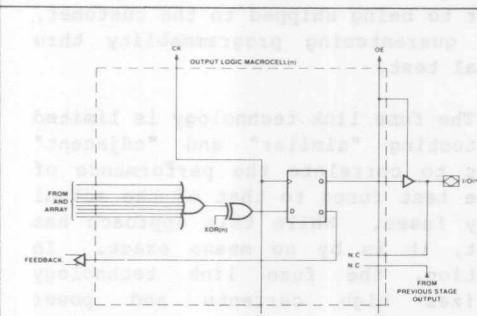
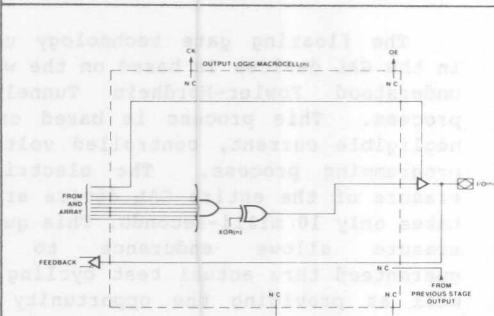
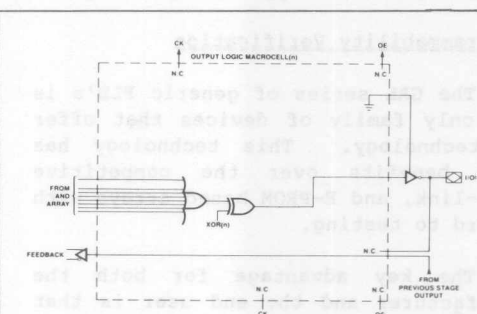
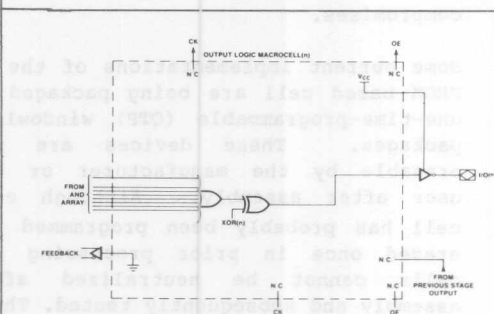
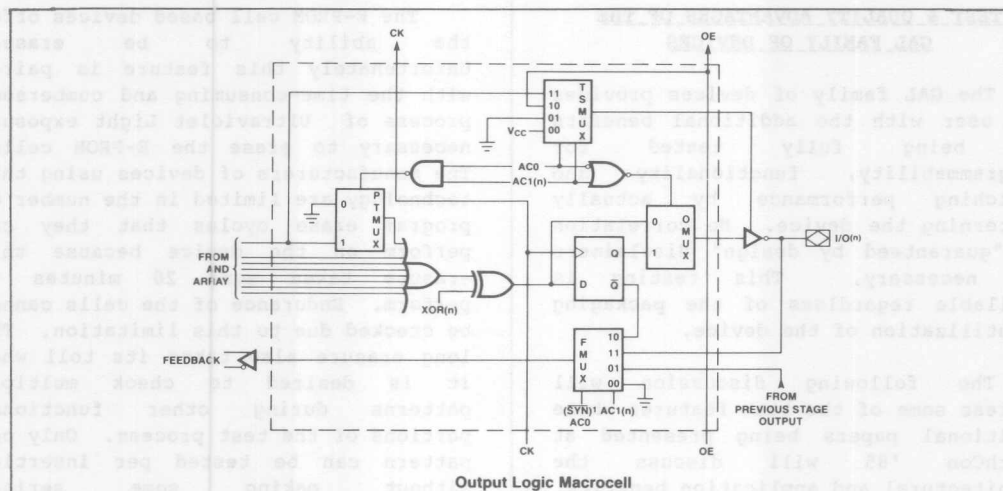
Product	Pins	Inputs	Outputs	Function-Polarity	Type
GAL-16V8	20	16	8	And-OR-Programmable	Configurable
RAL-10H8	20	10	8	And-OR	Asynchronous
RAL-10L8	20	10	8	And-OR-Invert	Asynchronous
RAL-10P8	20	10	8	And-OR-Programmable	Asynchronous
RAL-12H6	20	12	6	And-OR	Asynchronous
RAL-12L6	20	12	6	And-OR-Invert	Asynchronous
RAL-12P6	20	12	6	And-OR-Programmable	Asynchronous
RAL-14H4	20	14	4	And-OR	Asynchronous
RAL-14L4	20	14	4	And-OR-Invert	Asynchronous
RAL-14P4	20	14	4	And-OR-Programmable	Asynchronous
RAL-16H2	20	16	2	And-OR	Asynchronous
RAL-16L2	20	16	2	And-OR-Invert	Asynchronous
RAL-16P2	20	16	2	And-OR-Programmable	Asynchronous
RAL-16R8	20	16	8	And-OR	Synchronous
RAL-16RP8	20	16	8	And-OR-Programmable	Synchronous
RAL-16R6	20	16	6	And-OR	Synchronous
RAL-16RP6	20	16	6	And-OR-Programmable	Synchronous
RAL-16R4	20	16	4	And-OR	Synchronous
RAL-16RP4	20	16	4	And-OR-Programmable	Synchronous
RAL-16H8	20	16	8	And-OR	Asynchronous
RAL-16L8	20	16	8	And-OR-Invert	Asynchronous
RAL-16P8	20	16	8	And-OR-Programmable	Asynchronous



## LATTICE 24 PIN PRODUCTS

Product	Pins	Inputs	Outputs	Function-Polarity	Type
GAL-20V8	20	20	8	And-OR-Programmable	Configurable
RAL-14H8	20	14	8	And-OR	Asynchronous
RAL-14L8	20	14	8	And-OR-Invert	Asynchronous
RAL-14P8	20	14	8	And-OR-Programmable	Asynchronous
RAL-16H6	20	16	6	And-OR	Asynchronous
RAL-16L6	20	16	6	And-OR-Invert	Asynchronous
RAL-16P6	20	16	6	And-OR-Programmable	Asynchronous
RAL-18H4	20	18	4	And-OR	Asynchronous
RAL-18L4	20	18	4	And-OR-Invert	Asynchronous
RAL-18P4	20	18	4	And-OR-Programmable	Asynchronous
RAL-20H2	20	20	2	And-OR	Asynchronous
RAL-20L2	20	20	2	And-OR-Invert	Asynchronous
RAL-20P2	20	20	2	And-OR-Programmable	Asynchronous
RAL-20R8	20	20	3	And-OR	Synchronous
RAL-20RP8	20	20	3	And-OR-Programmable	Synchronous
RAL-20R6	20	20	5	And-OR	Synchronous
RAL-20RP6	20	20	5	And-OR-Programmable	Synchronous
RAL-20R4	20	20	4	And-OR	Synchronous
RAL-20RP4	20	20	4	And-OR-Programmable	Synchronous
RAL-20H8	20	20	8	And-OR	Asynchronous
RAL-20L8	20	20	8	And-OR-Invert	Asynchronous
RAL-20P8	20	20	8	And-OR-Programmable	Asynchronous





# TEST & QUALITY ADVANTAGES OF THE GAL FAMILY OF DEVICES

The GAL family of devices provides the user with the additional benefits of being fully tested for programmability, functionality and switching performance by actually patterning the device. No correlation or "guaranteed by design" disclaimers are necessary. This testing is available regardless of the packaging or utilization of the device.

The following discussion will address some of the test features while additional papers being presented at NorthCon '85 will discuss the architectural and application benefits.

## Programmability Verification

The GAL series of generic PLD's is the only family of devices that offer E<sup>2</sup> technology. This technology has many benefits over the competitive fuse-link, and E-PROM based arrays with regard to testing.

The key advantage for both the manufacturer and the end user is that the E<sup>2</sup> cell can be erased and re-used. This simple capability allows the cell to be programmed and erased many times prior to being shipped to the customer, thus guaranteeing programmability thru actual test.

The fuse link technology is limited to testing "similar" and "adjacent" fuses to correlate the performance of these test fuses to that of the actual array fuses. While this approach has merit, it is by no means exact. In addition, the fuse link technology utilizes high currents and power dissipation to accomplish the vaporization or pull-back of the metallic fuse links. This high current is routed around the chip thru large transistor and metal line structures which are subject to potential long term programming reliability problems. This is why the fuse-link algorithms require brief programming pulses with high voltage duty cycle constraints.

The E-PROM cell based devices offer the ability to be erased. Unfortunately this feature is paired with the time-consuming and cumbersome process of Ultraviolet Light exposure necessary to erase the E-PROM cells. The manufacturers of devices using this technology are limited in the number of program erase cycles that they can perform on the device because this erasure takes some 20 minutes to perform. Endurance of the cells cannot be checked due to this limitation. The long erasure also takes its toll when it is desired to check multiple patterns during other functional portions of the test process. Only one pattern can be tested per insertion without making some serious compromises.

Some current implementations of the E-PROM based cell are being packaged in one-time-programmable (OTP) windowless packages. These devices are not erasable by the manufacturer or the user after assembly. Although each cell has probably been programmed and erased once in prior processing the cells cannot be neutralized after assembly and subsequently tested. These manufacturers rely upon a correlation to a "phantom array"<sup>7</sup> for programmability and performance verification.

The floating gate technology used in the GAL devices is based on the well understood Fowler-Nordheim Tunneling process. This process is based on a negligible current, controlled voltage programming process. The electrical erasure of the entire GAL device array takes only 10 milli-seconds. This quick erasure allows endurance to be guaranteed thru actual test cycling as well as providing the opportunity to pattern the device many times during the manufacturing process to examine the actual functionality of the logic elements under direct control of the array cells exactly as the device would function under the control of a user defined pattern.

The  $E^2$  cell is larger than a corresponding E-PROM type cell. Historically this was a concern with PROM devices as they are die size limited by array cells, however, a programmable logic device is not composed of a lot of these cells. In fact, the GAL device array occupies only 6-7% of the total die area. The impact of the larger cell size on total die area is only 2-3%, a trivial amount to trade off for full testability by both the manufacturer and user. This tradeoff results in lower overall cost to the user since he no longer has to concern himself with (non-existent) reject devices.

The GAL programming algorithm utilizes a high speed serial shift scheme that allows simultaneous programming of 64 cells at one time. This means that the programming of the entire array takes 340 mS. This is considerably faster than the few seconds necessary for fuse link devices and the 10's of seconds necessary to program the UV erasable E-PLD devices.

All of the GAL family of devices also incorporates a proprietary **Margin Test** feature that allows the actual charge potential of each cell individually to be verified to ensure full programming margins. To verify retention all GAL devices are patterned and baked for 48 hrs to be sure that the individual cells retain their data. Weak cells that lose some, but not all, of their charge would not normally be detectable without this analog bit-by-bit margin test feature. The user of Lattice devices can have full confidence in the 10 year minimum retention specification of these devices, again due to the **actual test** performed on the cell. Currently approved programming hardware verifies that the cells are programmed well beyond the minimum margins required for proper retention and performance.

The GAL family of devices guarantees thru **actual test** the programability of each cell, its ability to retain charge for the specified time period, and its ability

to withstand repeated write/erase cycles (endurance). The impact of this "actual test" philosophy is directly observable in the quality levels of the GAL devices.

#### Switching & Functionality Verification

The verification that a patterned device performs as expected is no trivial task. Many man-years have gone into the development of test schemes for testing state machines without having to trade off confidence in functionality with cost of test. PLD's in the 20 and 24 pin families support a maximum of 10 registers to control the state of the system. This allows up to 1024 states; getting into each state to verify all next states is not a trivial task.

Typical approaches to verifying the functionality and performance of PLD devices include pseudo-random vector generation/signature analysis as well as the more traditional (and expensive) stuck at zero/stuck at one test vector set application. There are problems with both of these approaches that should be discussed.

The use of pseudo-random test vectors to test combinational logic is potentially a valid thing to do. The current implementation in relatively slow hardware does not test AC performance of the device and also takes several seconds to execute. This technique has several drawbacks when applied to sequential logic<sup>8</sup>. For example, consider a state machine with a common reset pin. This pin has a 50% chance of being reset during any given cycle. It is quite possible that many states may never be tested. The probability that a 10 bit (0 to 1023) counter would be reset in the first 1023 counts (a 50% chance each cycle) is near 100%. Validation that the counter would loop to state 0 after reaching 1023 would be next to impossible !!

Stuck at one (SA1)/stuck at zero (SA0) testing is based on the theories of observability and controllability of each node in a system. The basic theory indicates that if all nodes can be controlled and observed that proper logic functionality can be verified. Again, this is a valid concept for testing ideal systems where SA0 & SA1 are the only valid failure mechanisms. Semiconductors occasionally suffer from random defects that result in marginal or non-performance. CMOS technology can sometimes get stuck at the "last state" during some switching and defect induced fault conditions<sup>9</sup>. Stuck-at-Fault testing does not consider the potential degradation of the device performance due to random defects<sup>10</sup>.

There is hope in overcoming these difficulties and expenses. If the user could assume that all marginalities and functionalities had been previously eliminated from the devices under consideration a fuse verification would be sufficient to validate the device performance. Lattice feels that this is a valid alternative for the user of Lattice PLD's.

The GAL family of devices allows for generic testability of all functions under normal array control. The manufacturer (and user) can put any custom patterns they desire into the device for testing purposes, erase the device in 10 milliseconds and ultimately pattern the device with the final pattern with complete assurance that every piece of logic will work as specified. This testing can incorporate full switching time validation thru normal paths by **actual test**, not correlation.

Special voltages (super-voltages) and the temporary "loss" of a pin are not necessary to access the array (as with the fuse-link and phantom array UV erasable E-PLD devices). This leaves all device pins available to the user to validate functionality.

### Cost Effective Testing of GAL Devices

The GAL family of devices is tested under many different conditions during the manufacturing process. To minimize the test time during some of the early (wafer probe) test operations where functionality of various portions of the circuit are being verified for the first time it is to our advantage to test as quickly as possible. We have incorporated some special features into our device which allow the array to be overridden during the what we call the **Logic Test Mode**. Data form the serial-register-latch (SRL) overrides the array and travels thru the sense amps so that the subsequent logic circuitry is driven by the actual signals it will see in normal operation. This test mode is more comprehensive than some of the alternative approaches used in previous PLD design since the sense amp is not overridden. The sense amp is a critical untested portion the circuitry on other PLD's. The Logic Test Mode allows us to fully test the sense amps.

The functional testing of the output logic can be done in microseconds as opposed to milliseconds since it is not necessary to pattern the array to force a particular configuration or data condition. The cost savings of this technique over many test probes and insertions is dramatic. This manufacturing cost savings is passed along to the user in the form of higher quality devices with lower effective ASP devices.

The logic test mode is not used to test the switching performance of the device. Actual array and architecture programming is necessary to verify the switching performance under normal conditions. Some 10 to 12 different array patternings are performed during each of the various configurations of the device under worst case patterning conditions. Devices using fuse-link technology **must** use a correlated result and the E-FROM based devices are restricted to 1 or 2 compromised patterns due to the lack of a cost and time effective erase capability.



	GAL Devices	PAL Devices	UV Erasable E-PLD Devices	OTP E-PLD Devices
<b>Basic Specifications</b>				
Speed	25/35 ns	25/35 ns	35 ns or slower	25 ns
Maximum Active Current	90 mA	180/90 mA	50 mA	150 mA
Technology	CMOS, E <sup>2</sup>	Bipolar, Fuselink	CMOS, E-PROM	CMOS, E-PROM
Configurable Architecture	Yes	No	Yes	No
PAL Fuse Map Compatability	Yes	Yes	No	Yes
Devices to Supply Full Line	1	More than 20	1	Not Available
Electronic Signature	Yes	No	No	No
Security Cells	Yes	Yes	Yes	Yes

**Programming Issues**

Reprogrammable	Yes	No	Yes	No
Full Tested Fuse Array	Yes, Many Times	No	Yes, Once or Twice	Yes, Once
Test Fuse Array in Package	Yes	No	Yes	No
Endurance Test Capability	Yes	N/A	No	No
Programming Time	4 seconds	16 seconds	20 seconds	7 seconds
Erase Mechanism	Electrical	None	UV Lamp	None
Erase Time	10 Milliseconds	N/A	20 Minutes	N/A

**Packaged Test Issues**

Functionality Test	Actual Paths	Simulated	Actual Path	Phantom Path
Switching (AC) Test	Actual Paths	Correlated	Compromised Pattern	Correlated
Guaranteed Reliability	100% thru actual test	99% thru correlation	99% thru partial test	?? thru correlation



Testing A GAL Device In A System

The GAL family supports **register preload** to allow testing of sequential device configurations in a system or quickly on ATE. The current state can be quickly pre-loaded so that all of the various transitions to a next state can be tested. This feature is supported by the JEDEC standard data transfer protocol for test vector transfer. Many of the major device programmers also support this pre-load feature to functionally test a patterned PLD.

The implementation of preload in the GAL devices is somewhat unique in that it utilizes a serial shift scheme that allows both a load and unload of the registers. This serial-in serial-out shift technique allows the system designer or test engineer to devise a daisy-chaining scheme for forcing data. This scheme allows 100% controllability and observability of the device states.

SUMMARY

The GAL-16V8 and the GAL-20V8 devices and their family members bring a new level of flexibility and quality to the user of programmable logic. The generic approach to functionality and compatibility and the philosophy of guaranteeing performance thru actual test makes the GAL family of devices the ideal choice for both prototyping and volume production environments.

ACKNOWLEDGMENTS

The author would like to recognize all of the individuals responsible for the Circuit and Topological Design, Process Architecture, Silicon Manufacture, Product Engineering, and Test Engineering that make the GAL concept possible.

LEGALITIES

GAL, Generic Array Logic, RAL, Reprogrammable Array Logic, ULTRAMOS and Silicon Forest are trademarks of Lattice Semiconductor Corporation.

- 1 Sievers, William H.  
Advanced Micro Devices  
"How Testability Is Designed Into Programmable Logic", pg. 10/2  
MidCon '84, Dallas, Texas.
- 2 Registered trademark of Monolithic Memories Inc.
- 3 Trademark of Monolithic Memories Inc.
- 4 Trademark of Assisted Technology Corporation.
- 5 Trademark of Data I/O Corporation.
- 6 Trademark of Stag Microsystems Inc.
- 7 Elliot, Dane  
Cypress Semiconductor  
NorthCon '84 Presentation.
- 8 James, George W.  
Data I/O Inc.  
"Testing is a Key Element In the Successful Use of PROM's and PLD's"  
Evaluation Engineering, pg. 95  
July 1985.
- 9 Reddy, Sudhakar M.  
University of Iowa  
"On CMOS Totally-Self-Checking Checkers"  
SRC Topical Research Conference--  
Built In Test/Testability, pg.10
- 10 Graf, Matthew C.  
IBM East Fishkill, NY  
"Testing- A major Concern for VLSI"  
Solid State Technology, pg. 101

DESIGN METHODOLOGY FOR NEW PROGRAMMABLE LOGIC DEVICES (1985)  
Investigating new design tools for VLSI logic devices

Lattice Semiconductor Corporation  
12500 NE Greenway Parkway  
Sunnyvale, CA 94086  
408 239-1121

Using Verilog and Synopsys's  
theories to make the system logic fit in  
a target device. PALM is limited to  
PAL devices only, and does not  
support PROM or EPROM.  
New high-level hardware programming  
languages have been developed that are

INTRODUCTION  
Many design engineers have been  
reluctant to turn to programmable logic  
devices (PLDs) for new system designs  
because logic systems for PLDs were  
designed with minimum run-of-product  
logic systems in mind, rather than the

LATTICE

## DESIGN METHODOLOGY FOR NEW PROGRAMMABLE LOGIC DEVICES

that provide system abstraction, logic  
simulation, and design realization. With  
these new software tools, a design can  
be specified with any combination of  
logic functions, boolean equations, or  
truth tables. A design engineer chooses  
the hardware that best fits a  
particular logic design.

These new languages remove the need of  
a designer to understand the  
underlying hardware. Particularly when  
coupled with one of the new generic  
array logic products which feature  
variable architecture, programmable  
logic suddenly becomes very attractive  
for logic design. Designers can be  
freed from the constraints of a logical,  
functional manner, and described their  
way in a high level programming  
language. Designers can now specify  
logic devices can be developed, tested,  
and simulated without concern about  
which logic or memory cells will be  
programmed to implement the logic or  
device architecture.

In the balance of this paper, the  
features and benefits of PALM and CHDL  
will be viewed, along with a  
description of new variables  
architecture TPL devices, the PALM  
and the CHDL. Some design rules for  
the new tools will be presented, and the  
new conclusions will look into  
the future of programmable logic  
design.

### THE NEW PROGRAMMABLE LANGUAGE

The syntax of PALM resembles the C  
software programming language. As a  
result, designs are easy to read and  
understand. Figure 1 is an example from  
the PALM application. The first

and syntax of a computer or assembly  
language makes logic design, and  
design, or even when a designer  
wanted TPL designs, the logic didn't  
quite fit in available PLDs, and extra  
effort was expended on reconfiguring  
or reconfiguring the logic design.

Now, second generation silicon devices  
feature high performance CMOS  
technology with reprogrammable memory  
cells and configurable architectures  
provide a universal device solution.  
For the first time, the silicon can be  
tailored to fit the logic, rather than  
the other way around. This paper will  
describe how to design systems with  
these new reprogrammable logic devices,  
using the latest hardware and software  
development tools.

### THE NEW SOFTWARE TOOLS

The most popular of the early software  
development tools is PALASM (Lattice  
and Coll, 1987). PALASM is a simple  
language that converts boolean  
equations, which represent the standard  
functions from device input to output,  
into LUTEC standard form. The logic  
is used to build or program PLDs. With  
this prior generation tool, the logic  
equations must be reduced manually

PAL is a registered trademark of  
Monolithic Memories, Inc., Santa Clara,  
Ca.

DESIGN METHODOLOGY FOR NEW PROGRAMMABLE LOGIC DEVICES (PLDs)  
Leveraging new design tools for PLDs into instant custom silicon

Lattice Semiconductor Corporation  
15400 NW Greenbrier Parkway  
Beaverton, OR. 97006  
503 629-2131

## INTRODUCTION

Many design engineers have been reluctant to turn to programmable logic devices (PLDs) for new system designs because logic systems for PLDs were designed with minimized sum-of-products form boolean equations rather than the familiar schematic diagrams. Those that braved the transition faced a myriad of obstacles, including learning a computer operating system, a full-screen text editor, the language and syntax of a compiler or assembly language program, boolean theory, and the various architectures of programmable logic devices. The lack of reasonable hardware and software support was the biggest obstacle of all. All too often, after having carefully chosen a target PLD for a design, or even when converting existing TTL designs, the logic didn't quite fit in available PLDs, and extra effort was expended on repartitioning or respecifying the logic designs.

Now, second generation silicon devices featuring high performance CMOS technology with reprogrammable memory cells and configurable architecture provide a universal device solution. For the first time, the silicon can be tailored to fit the logic, rather than the other way around. This paper will describe how to design systems with these new reprogrammable logic devices, using the latest hardware and software development tools.

## THE NEW SOFTWARE TOOLS

The most popular of the early software development tools is PALASM (Birkner and Coli, 1983). PALASM is a simple language that converts Boolean equations, which represent the transfer functions from device input to output, into JEDEC standard fuse map files that are used to burn or program PLDs. With this prior generation tool, the logic equations must be reduced manually

PAL is a registered trademark of Monolithic Memories, Inc., Santa Clara Ca.

using Karnaugh maps and DeMorgan's theorem to make the system logic fit in a target device. PALASM is limited to PAL<sup>R</sup> devices only, and does not support PROMs or FPLAs.

New high-level hardware programming languages have been developed that are useful for defining logic designs for a variety of PLDs. These include ABEL, from DATA/IO in Redmond, Wa. and CUPL, from Assisted Technology, in San Jose, Ca. Both are compiler-based software that provide syntax checking, logic reduction, and design simulation. With these new software tools, a design can be specified with any combination of state diagrams, boolean equations, or truth tables. A design engineer chooses the construct that best fits a particular logic design.

These new languages remove the need of a designer to understand PLD architectures. Particularly when coupled with one of the new generic array logic products which feature variable architecture, programmable logic suddenly becomes very attractive for logic design. Designs can be thought through in a logical, functional manner, and described that way in a high level programming language. Designs using generic array logic devices can be developed, tested and simulated without concern about which fuses or memory cells will be programmed to implement the logic or device architecture.

In the balance of this paper, the features and benefits of ABEL and CUPL will be viewed, along with a description of new variable architecture PLD devices, the GAL16V8 and the GAL20V8. Some design rules for the new PLDs will be presented, and the paper concludes with brief look into the future of programmable logic design.

## THE ABEL PROGRAMMING LANGUAGE

The syntax of ABEL resembles the C software programming language. As a result, designs are easy to read and understand. Figure 1 is an example from the ABEL applications guide. The truth



The SIMULATE program takes the intermediate file output from the FUSEMAP program, including test vectors, along with a device specification file provided with ABEL to fully simulate the actual operation of a device design. The fuse map and test vectors used in device simulation are the same as those used to program and test the real PLD. An iterative process is used to simulate both combinational and registered configurations (with feedback), so that unstable designs can be detected.

Finally, the DOCUMENT program uses intermediate output files from the other programs to create a text file that fully documents the design. Options in the document include original equations and those produced from truth tables and state diagrams, transformed and reduced equations, a fuse map pictorial, a symbol table, test vectors, and a pin diagram.

ABEL is a very natural and complete logic design software tool for all forms of programmable logic. The high level language is combined with various levels of automatic logic reduction, full design simulation, error checking, and design verification through test vectors.

ABEL runs under MStm-DOS or PC-DOS<sup>R</sup> operating systems on the IBM<sup>R</sup> PC and most compatibles, or on VAXtm systems under VMStm 3.0 or Berkeley UNIXtm version 4.2.

#### CUPL - A UNIVERSAL HARDWARE COMPILER

CUPL<sup>tm</sup> is an industry standard high-level programming language (from Assisted Technology, San Jose, Ca.). CUPL is, like ABEL, compiler-based, and supports almost all PLD devices, including PROMs and FPLAs. The commitment to support all types of programmable devices is important to system designers because it provides them a wealth of devices to choose from. This is in contrast to the assembler-based software available from PLD manufacturers, which typically support only one family of devices from a given manufacturer.

With CUPL, designers can avoid sole source situations. As shown in figure 3, a single logic description in CUPL can be compiled into several different devices from many different companies, providing multiple sources for system sockets.

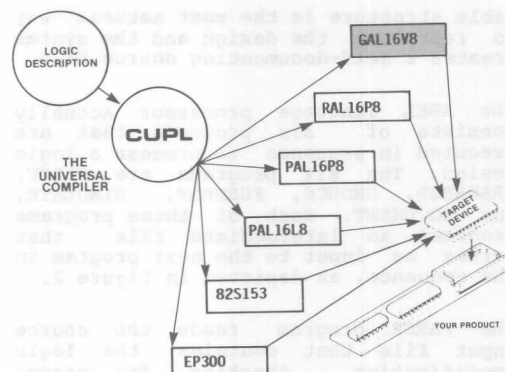


Figure 3 Multiple sources at the socket

The syntax of CUPL is chosen from the C language as follows:

& = logical AND

# = logical OR

\$ = logical exclusive-OR

! = logical inverse (negation)

Comments can be placed anywhere in the source code and are enclosed in the familiar /\* ...comments ... \*/ form. Symbolic names can be used to define pin variable names, internal nodes, intermediate variables, constants and bit-fields. Macro substitution and bit-field specifications are two of the major advantages that CUPL (and ABEL) have over assembler-based languages. This is readily apparent upon examination of figure 4.

#### Comparative Design Example:

Produce an I/O Buffer Enable signal which is asserted over the Hex Address Range 10 thru 14.

#### Equations Using Boolean Assembler/Translator

BUFFEN =

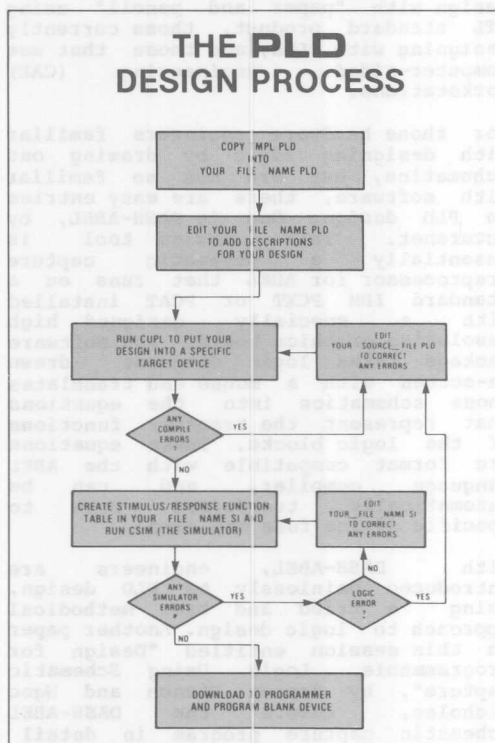
$$\begin{aligned} & \text{IORD} \cdot \text{A7} \cdot \text{A6} \cdot \text{A5} \cdot \text{A4} \cdot \text{A3} \cdot \text{A2} \cdot \text{A1} \cdot \text{A0} \\ & + \text{IOWR} \cdot \text{A7} \cdot \text{A6} \cdot \text{A5} \cdot \text{A4} \cdot \text{A3} \cdot \text{A2} \cdot \text{A1} \cdot \text{A0} \\ & + \text{IORD} \cdot \text{A7} \cdot \text{A6} \cdot \text{A5} \cdot \text{A4} \cdot \text{A3} \cdot \text{A2} \cdot \text{A1} \cdot \text{A0} \\ & + \text{IOWR} \cdot \text{A7} \cdot \text{A6} \cdot \text{A5} \cdot \text{A4} \cdot \text{A3} \cdot \text{A2} \cdot \text{A1} \cdot \text{A0} \\ & + \text{IORD} \cdot \text{A7} \cdot \text{A6} \cdot \text{A5} \cdot \text{A4} \cdot \text{A3} \cdot \text{A2} \cdot \text{A1} \cdot \text{A0} \\ & + \text{IOWR} \cdot \text{A7} \cdot \text{A6} \cdot \text{A5} \cdot \text{A4} \cdot \text{A3} \cdot \text{A2} \cdot \text{A1} \cdot \text{A0} \end{aligned}$$

#### Equations Using CUPL:

Intermediate Equations { Bit Field = FIELD ADR = [A7..0]; List Notation  
Expression Substitution > IOREQ = IORD # IOWR;  
Final Logic Equation Uses Intermediate > BUFFEN = IOREQ & ADR[10..14]; Range Operation



The PLD design process with CUPL is straight forward, and is diagrammed in figure 5. The CUPL software package provides a skeletal source file as a suggested starting point for creating a specific design source. Once the design is completed (source code can be expressed in truth table form, high level boolean equations or state machine syntax) the source is compiled with the target device specified in the compile command line, along with the level of logic reduction required (none, quick, reduce until fits, and



full minimization). The compiler has several output options including list and documentation files, as well as JEDEC formatted output for downloading to a programmer.

The second half of the CUPL design process is running the logic simulator, CSIM. The CSIM simulator takes a logic description from the compiler and combines it with a stimulus/response function table written by the design or evaluation engineer, and performs a design simulation. CSIM will output a text file that specifies any errors, and a JEDEC fuse map file that includes test vectors.

Designed for hardware engineers, CUPL is an easy to learn, simple, powerful and self-documenting language useful for any programmable logic device design. The development vehicle can be any CP/M, MSDOS, UNIX, or VMS based computer.

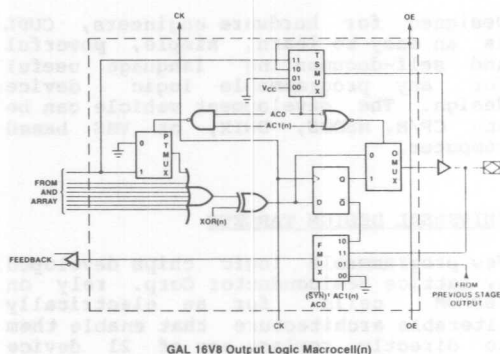
#### UNIVERSAL DESIGN TARGETS

New programmable logic chips developed by Lattice Semiconductor Corp. rely on EEPROM cells for an electrically alterable architecture that enable them to directly replace any of 21 device architectures in the 20-pin and 24-pin bipolar PAL family manufactured by Monolithic Memories Inc. Called the GAL™ 16V8 (for Generic Array Logic), and the GAL 20V8, Lattice's low-power CMOS devices offer bipolar speed (25-nanosecond propagation delay) and output drive (24-milliampere IOL), but replaces the usual fuses with EEPROM cells. The result is electrically reprogrammable and reusable devices that can be programmed on standard logic-array programmers, and can use the new standard software (such as ABEL and CUPL).

Lattice is the first company to apply EEPROM technology to a programmable logic device. The new user benefits are numerous. GAL devices can be used over and over during system prototyping; a device once used as a 14L8, for example, can be reprogrammed as a 20R6 for another circuit. Thanks to the advanced output logic architecture, each output can be individually tailored as registered, combinational, active-high, active-low, input-only, or bi-directional. This generic architectural flexibility relieves the design engineer from being forced to choose a target device for his logic design ahead of time. By removing this design encumbrance, many more design engineers will opt for programmable logic devices in their designs.

The key element of the GAL 16V8 that provides its architectural flexibility is the output logic macrocell (OLMC), shown in Fig. 6. The OLMC allows each output to be individually set to the system designer's choice of active high or active low, with either combinational (asynchronous) or registered (synchronous) configurations. A common output enable can be connected to all outputs, or product terms can be used to provide individual output enable control. The output logic macrocell provides the designer with maximum output flexibility in matching signal requirements, thus providing a universal target for all possible 20-pin logic device designs.





### Beyond Ordinary PLD devices

The GAL 16V8 is a superb replacement for the bipolar family of 20-pin devices; however, the flexibility of the output logic macrocell (OLMC) allows the GAL devices to be configured in many ways that are not possible with ordinary PLD devices. For example, the ability to place a synchronous (registered) output on any or all of the output pins allows the user to create a variety of devices such as a 16R1 or a 16R5 -- architectures not possible with the ordinary PLD devices available today.

The need for "odd" architectures often arises during the system design process. For example, suppose the target device for a latched two-to-four decoder is a 16R4, a reasonable choice. If one of the decoded functions turns out to need a "not-Ready" response, requiring one extra state bit for wait-state insertion, a designer must add some standard TTL glue circuitry to include a wait-state generator, or search through catalogs to find another device.

Unfortunately, no ordinary PLD device includes only five registers, and in such circumstances, the 16V8 device excels. Using the GAL 16V8 as the target device with one of the available software tools, such as CUPL, in fact, turns the situation around; for the first time, the software will tailor the silicon to fit the design, rather than forcing an engineer to respecify or repartition his logic to fit a device that "almost" meets the design requirements.

This more efficient packing of logic into the GAL 16V8 will allow designers to increase functional density over that currently possible with PLD-based designs. The increase in density may

range from 5 to 25% based on the current usage of PLD devices in systems.

A key point here is that the ability to use these more unconventional configurations of the GAL 16V8 is fully supported by the industry standard PLD software support packages, ABEL and CUPL.

### DESIGN METHODS

There are three groups of design engineers who can benefit from the new software tools and silicon. Those that design with "paper and pencil" using TTL standard product, those currently designing with PLDs, and those that use computer-aided engineering (CAE) workstations.

For those hardware engineers familiar with designing logic by drawing out schematics, but are not so familiar with software, there are easy entries to PLD design. One is DASH-ABEL, by Futurenet. This design tool is essentially a schematic capture preprocessor for ABEL that runs on a standard IBM PCXT or PCAT installed with a specially designed high resolution graphics board. The software package takes logic diagrams drawn on-screen with a mouse and translates those schematics into the equations that represent the transfer functions of the logic blocks. These equations are format compatible with the ABEL language compiler, and can be automatically transformed down to specific device fuse map data.

With DASH-ABEL, engineers are introduced painlessly to PLD design, using a tried and true methodical approach to logic design. Another paper in this session entitled "Design for Programmable Logic Using Schematic Capture", by Brenda French and Ngoc Nicholas, covers the DASH-ABEL schematic capture program in detail. CUPL-GTS by Assisted Technology is another example of a schematic capture preprocessor; in this case the preprocessor incorporates CUPL in its internal operation, providing the benefits of automatic logic reduction, and simulation through CSIM. Once a logic schematic has been entered, an engineer can quickly check to determine if the logic fits in a target device. Through pop-up menus, the entire design process can be handled with the mouse; keyboard entry and programming is not required.

System designers using CAE workstations can now benefit from the development of the high level PLD design software.

Until now, engineers designing logic on workstations had been limited to semicustom or full custom logic design solutions. However, as CAE workstations migrate to board-level system design capability, PLD design capability is being offered. The effort to link workstations with PLD design software is being lead by Assisted Technology through its recent merger with PCAD of Los Gatos, CA.

Until recently, PLDs were treated as black holes during system simulation on workstations. CUPL now has an interface to workstation software, thus providing device-dependent information to workstation software. It is important to use this capability during system simulation, so that an accurate device model is presented to the workstation software, including the necessary minimization of the PLD design.

Recall that devices with logic macrocells can have pins that are outputs in one design and inputs in another. This can causes problems in extracting a device image from the workstation shape library, unless a proper interface is provided by the PLD software. Correct interpretation of architectural configuration data for correct topological modeling is provided by using CUPL with workstation software. As new PLD devices continue to move into the gate array market, the traditional home-ground of workstations, the recently forged links between workstation software and PLD software will play an increasingly important role in the continued growth of the PLD business.

The last group of engineers to benefit from the emergence of the new PLD design tools are those that are currently designing with PLD devices. For those engineers, the benefits will become readily apparent after the first design cycle using the new tools. To take best advantage of the new software tools, get familiar with all the capabilities of the software. In the migration from assembler based software to the new tools, old habits are easy to fall back on. Take advantage of the ability to create documentation within the source file. Use bit field notation and expression substitution where appropriate to simplify the specification and increase the clarity of the document.

When designing state machines, use identifiers rather than numbers for states and state transitions. Take advantage of the power-up reset capability of newer PLDs in setting the machine to a known state on power-up. This can save product terms for another purpose (sometimes this is not possible

if the logic requires reset capability for other than power-up).

With ABEL, unsatisfied state transition conditions clear the state register. This can be used to some advantage to eliminate product term usage explicit definition of all transition. Keep in mind, though, a state must always be assigned to the cleared-register state.

Another good idea is to number adjacent states for one-bit changes (the state diagram maps to an N-cube). Select numeric constants for the state names so that state registers change by one bit only (ie. grey code transitions) from one state to the next. This can greatly reduce the number of product terms required to describe state transitions, and is a step in providing hazard-free logic designs.

Finally, a word of caution when creating creating hazard-free logic designs. Often, a redundant term is introduced into a logic expression to produce "glitch-free" outputs during input transitions of combinational networks. This is a proven and valid logic design technique. (Refer to Harris Programmable Logic Application note 106, "Hazard Free Logic Design", by Steve Bennett). Unfortunately, when using either of the automatic reduction algorithms built-in to CUPL or ABEL, all redundant terms are eliminated, whether the redundancy is intentional or not. Be aware that certain levels of logic reduction can potentially recreate a logic hazard in the design. This is easily handled by using a simpler level of reduction, where redundant terms are not eliminated, and can be used safely.

#### REFERENCES

1. ABEL 1.1 Applications Guide, DATA I/O Corp, 1985
2. CUPL User's Manual, Assisted Technology, 1984
3. Kyu Y. Lee, and others, "A High-Level Language for Programmable Logic Devices", VLSI Design, June 1985
4. Osann, Bob, "Compiler-based software and PLDs improve logic design", EDN, January, 1985
5. Osann, Bob, "Handling Programmable Logic on CAE Workstations", VLSI Design, July 1985

It is logic register test capability  
to allow them power-up.

With AEL, unswitched state transition  
conditions clear the state register.  
This can be used to some advantage to  
eliminate spurious state changes  
caused by all transitions. Keep in  
mind, though, a state must always be  
assigned to the cleared-register state.

A further good idea is to number adjacent  
states for one-bit changes (the state  
transition logic is an N-tuple). Defect  
analysis compares the state names  
to that state register change by one  
bit only (a Gray code transition).  
This one state in the next. This can  
greatly reduce the number of product  
forms required to describe the state  
transitions, and is a step in providing  
hardware logic designers.

Finally, a word of caution when  
creating existing hardware logic  
designs. Often, a redundant term is  
introduced into a logic expression to  
provide "glitch-free" outputs during  
input transitions of combinational  
networks. This is a proven and valid  
logic design technique. Refer to  
Lattice Programmable Logic Application  
Note 101, "Preventing Logic Design",  
by Steve Bennett. Unfortunately, when  
using either of the automatic conversion  
algorithms built-in to CUPL or AEL,  
all redundant terms are eliminated.  
The redundancy is introduced in  
order to meet the needs of certain  
logic reduction can potentially  
introduce a logic hazard in the design.  
This is easily handled by using a  
higher level of reduction, where  
redundant terms are not eliminated, and  
can be easily verified.

# REFERENCES

1. Lattice Application Guide,  
DATA 7-0 Corp, 1985
2. CUPL User's Manual, Lattice  
Technology, 1984
3. Kuo, L. Lee, and Robert, "A  
High-Level Language for Programmable  
Logic Devices", First Design, June  
1985
4. Thomas, Bob, "Compiler-based software  
and high-level logic design", ESD:  
January, 1985
5. Cohen, Bob, "Modeling Programmable  
Logic on CAD Packages", VLSI  
Design, July 1985

Until now, engineers designing logic on  
workstations had been limited to  
simulation or full custom logic design  
solutions. However, as CAD workstations  
migrate to desktop-level system design  
capability, ESD design capability is  
being altered. The effort to limit  
workstations with ESD design capability  
is being led by Lattice Technology  
through its recent merger with ESD of  
Los Gatos, CA.

Until recently, ESDs were restricted to  
black boxes during system simulation on  
workstations. CUPL now has an interface  
to workstation software. This provides  
device-dependent information to  
workstation software. It is important  
to use this capability during system  
simulation, to find an accurate device  
model is provided to the workstation  
software, including the necessary  
simulation of the ESD design.

Finally, that devices with logic  
capabilities can have pins that are  
outputs in one design and inputs in  
another. This can cause problems in  
creating a device model from the  
workstation design library, unless a  
proper interface is provided by the  
PLD software. Correct interpretation of  
external configuration data for  
external logical modeling is  
provided by using CUPL with workstation  
software. As new PLD device capabilities  
move into the gate array market, the  
traditional hardware-oriented  
workstations, the recently adapted logic  
between workstation software and PLD  
software will play an increasingly  
important role in the continued growth  
of the PLD business.

The last group of engineers to benefit  
from the emergence of the new PLD  
design tools are those who are  
currently designing with PLD devices.  
For those engineers, the benefits will  
become readily apparent after the first  
design cycle using the new tools. To  
take best advantage of the new software  
tools, get familiar with all the  
capabilities of the software. To the  
maximum, use assembly-based software  
for the new tools, old habits are easy  
to fall back on. Take advantage of the  
ability to create documentation within  
the source file. Use bit-level notation  
and expansion notation where  
appropriate to simplify the  
specification and increase the clarity  
of the document.

When designing state machines, use  
identifiers rather than numbers for  
states and state transitions. Take  
advantage of the power-up reset  
capability of newer ESDs in setting the  
machine to a known state on power-up.  
This can save product costs for another  
purpose. Sometimes this is not possible

## E<sup>2</sup>CMOS PROGRAMMABLE LOGIC: SUPERIOR QUALITY, FLEXIBILITY AND SYSTEM PERFORMANCE

Jerry Greiner  
Applications Manager  
LATTICE SEMICONDUCTOR CORP.  
13400 SW Greenway Parkway  
Hawthorne, Oregon 97086  
(503) 423-1111

are the 20-pin GAL<sup>TM</sup>16VLSI (static CMOS logic) and 24-pin GAL16VLSI. The technology in general and the GAL16VLSI in particular will be examined in the forthcoming article.

THESE TWO GAL16VLSI

Lattice Semiconductor is pioneering the use of Electrically Erasable (E<sup>2</sup>) CMOS technology to build reconfigurable logic devices. Low power programmable logic devices (PLDs) and PLDs have been developed utilizing 1.5 micron CMOS floating gate memory technology and unique design methodology to

## E<sup>2</sup>CMOS PROGRAMMABLE LOGIC: SUPERIOR QUALITY, FLEXIBILITY AND SYSTEM PERFORMANCE

Jerry Greiner

This paper was presented at Electro '86, May 14, in Boston, Massachusetts, and appeared in the Electro/86 Professional Session Record, Session 12, Copyright © Electronic Conventions Management, Inc. □

performance, the array has been split into two halves with row drivers driving both directions (Figure 2). This effectively halves the propagation delay line and leaves the row to a device which needs a maximum propagation delay of just 15ns.

A novel, high speed programming technique has been implemented through the use of an 8-bit serial shift register. This allows an entire row of data (64 bits) to be serially loaded into the device and subsequently parallel programmed onto a selected row in one programming cycle. Since the entire array contains a total of 40 rows, the device can be completely programmed in 40ms (40 rows x 10ms).

The GAL16VLSI is packaged in a 20-pin DIP and features two inputs on pins 1-11 and eight bi-directional pins 12-19 and output logic Macrocells (CMOS) on pins 20-23. Configuration is achieved through the OLMC, which is shown in Figure 3. Four different switching modes to allow any of the architecture configurations shown in Figures 4-7. The designer can choose to configure each OLMC as either a dedicated input, dedicated output, bi-directional output, or dedicated output with feedback.

Advantages of high functional speed at the expense of relatively high power consumption and lack of flexibility (one-time programmability). Another major flexibility has only been available through hardware mask options. Recent developments in digital technology have produced one-time field configurable architectures. CMOS floating gate technologies (FPGA/EPROM) have been utilized to address the reprogrammability and reconfigurability issues but these devices typically utilized crosspoint speed (input/output) in order to achieve flexibility.

Lattice has chosen this hardware by introducing CMOS FPGAs implemented in a high performance floating gate process which approaches digital performance levels at significantly reduced power levels. The E<sup>2</sup> memory technology has also been utilized to realize a device that is not only reprogrammable but also architecturally reconfigurable. The first two products in the family

E<sup>2</sup>CMOS PROGRAMMABLE LOGIC: SUPERIOR QUALITY, FLEXIBILITY,  
AND SYSTEM PERFORMANCE

Jerry Greiner  
Applications Manager  
LATTICE SEMICONDUCTOR CORP.  
15400 NW Greenbrier Parkway  
Beaverton, Oregon 97006  
(503) 629-2131

Lattice Semiconductor is pioneering the use of Electrically Erasable (E<sup>2</sup>) CMOS technology to build reconfigurable, high speed, low power programmable logic devices (PLD). A 25ns PLD has been developed utilizing 1.5 micron CMOS floating gate memory technology and unique design methodology to achieve high performance and superior quality. The technology and design approach will be examined in this paper.

#### INTRODUCTION

Over the past ten years, a number of programmable logic architectures and devices have been developed. The majority of the commercially successful PLDs have implemented in bipolar technology. This technology has the advantage of high functional speed at the expense of relatively high power consumption and lack of flexibility (one-time programmability). Architectural flexibility has only been available through hardwired mask options. Recent developments in bipolar technology have produced one-time field configurable architectures. MOS floating gate technologies (EPROM/EEPROM) have been utilized to address the reprogrammability and reconfigurability issues but these devices typically suffered tremendous speed impact (100ns) in order to achieve flexibility.

Lattice has cleared this hurdle by introducing CMOS PLDs implemented in a high performance floating gate process which approaches bipolar performance levels at significantly reduced power levels. The E<sup>2</sup> memory technology has also been utilized to realize a device that is not only reprogrammable but also architecturally reconfigurable. The first two products in the family

are the 20-pin GAL<sup>TM</sup>16V8 (Generic Array Logic) and 24-pin GAL20V8. The technology in general and the GAL20V8 in particular will be examined in the forthcoming text.

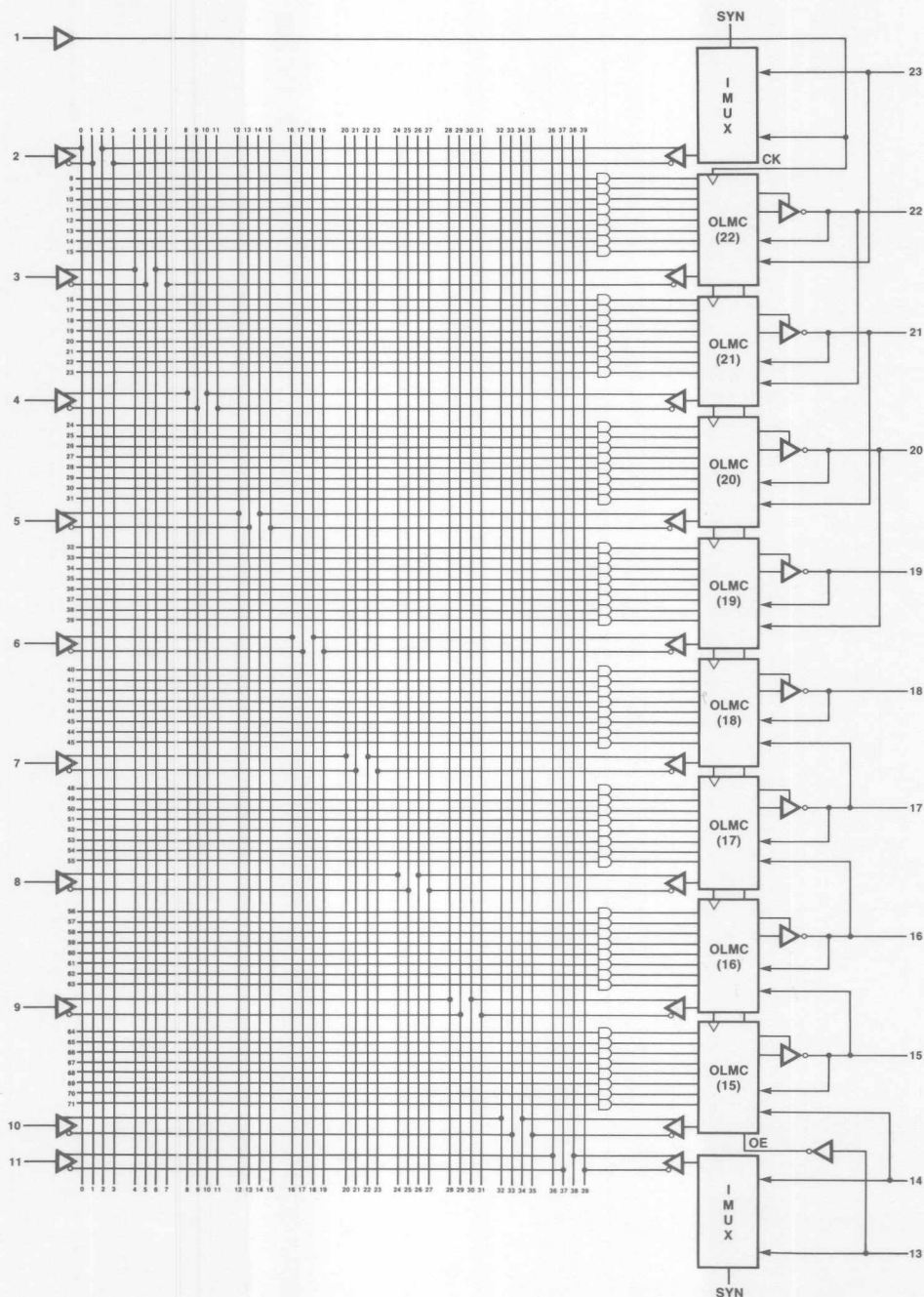
#### INSIDE THE GAL20V8

The logic diagram of the reconfigurable GAL20V8 is shown in Figure 1. The corresponding die photograph is shown in Figure 2. To achieve higher performance, the array has been split into two halves with row drivers driving both directions (Figure 2). This effectively halves the polysilicon row delay line and paves the road to a device which touts a maximum propagation delay of just 25ns.

A novel, high speed programming technique has been implemented through the use of an 82-bit serial shift register. This allows an entire row of data (64 bits) to be serially loaded into the device and subsequently parallel programmed onto a selected row in one 10ms programming cycle. Since the entire array contains a total of 40 rows, the device can be completely programmed in 400ms (40 rows x 10ms).

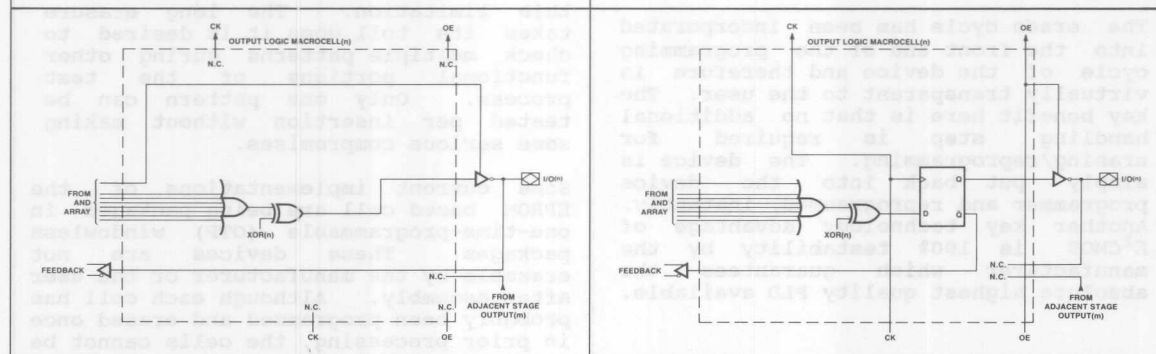
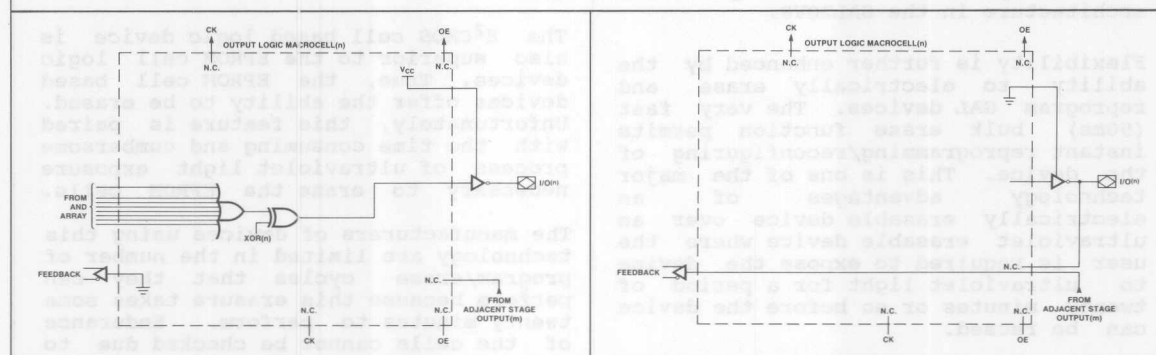
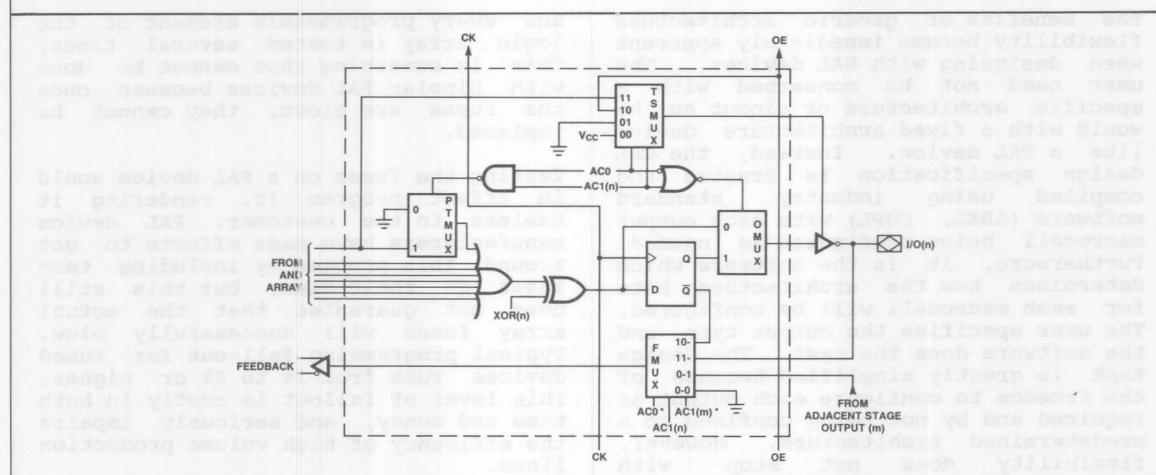
The GAL20V8 is packaged in a 300 mil 24-pin DIP and features ten inputs on pins 2-11 and eight bi-directional programmable Output Logic Macrocells (OLMC) on pins 15-22. Architecture configuration is achieved through the OLMC, which is shown in Figure 3. Four different multiplexers combine to allow any of the architecture configurations shown in Figures 4-7. The designer can choose to configure each OLMC as either a dedicated input, dedicated combinational output, bi-directional combinational output with feedback, or registered output with feedback.











The benefits of generic architecture flexibility become immediately apparent when designing with GAL devices. The user need not be concerned with a specific architecture or pinout as he would with a fixed architecture device like a PAL device. Instead, the GAL design specification is created and compiled using industry standard software (ABEL, CUPL) with each output macrocell being configured as needed. Furthermore, it is the software which determines how the architecture bits for each macrocell will be configured. The user specifies the output type and the software does the rest. The design task is greatly simplified because of the freedom to configure each output as required and by not being confined to a predetermined architecture. However, flexibility does not stop with architecture in the GAL20V8.

Flexibility is further enhanced by the ability to electrically erase and reprogram GAL devices. The very fast (50ms) bulk erase function permits instant reprogramming/reconfiguring of the device. This is one of the major technology advantages of an electrically erasable device over an ultraviolet erasable device where the user is required to expose the device to ultraviolet light for a period of twenty minutes or so before the device can be reused.

The erase cycle has been incorporated into the front end of the programming cycle of the device and therefore is virtually transparent to the user. The key benefit here is that no additional handling step is required for erasing/reprogramming. The device is simply put back into the device programmer and reprogrammed, instantly. Another key technology advantage of E<sup>2</sup>CMOS is 100% testability by the manufacturer which guarantees the absolute highest quality PLD available.

#### TESTABILITY: E<sup>2</sup>CMOS DOMINATES COMPETING TECHNOLOGIES

The GAL family are the only devices that offer E<sup>2</sup>CMOS technology. This technology has many benefits over the competitive fuse-link, and EPROM (ultra-violet) based arrays with regard to testing. With the EEPROM based programming, a GAL device is completely tested before it ever leaves the factory, including programming. Each

and every programmable element of the logic array is tested several times. This is something that cannot be done with bipolar PAL devices because once the fuses are blown, they cannot be replaced.

Testing the fuses on a PAL device would in effect program it, rendering it useless to the customer. PAL device manufacturers have made efforts to get around this problem by including test fuses on their dies, but this still does not guarantee that the actual array fuses will successfully blow. Typical programming fall-out for fused devices runs from 1% to 2% or higher. This level of fallout is costly in both time and money, and seriously impairs the efficiency of high volume production lines.

The E<sup>2</sup>CMOS cell based logic device is also superior to the EPROM cell logic devices. True, the EPROM cell based devices offer the ability to be erased. Unfortunately, this feature is paired with the time consuming and cumbersome process of ultraviolet light exposure necessary to erase the EPROM cells.

The manufacturers of devices using this technology are limited in the number of program/erase cycles that they can perform because this erasure takes some twenty minutes to perform. Endurance of the cells cannot be checked due to this limitation. The long erasure takes its toll when it is desired to check multiple patterns during other functional portions of the test process. Only one pattern can be tested per insertion without making some serious compromises.

Some current implementations of the EPROM based cell are being packaged in one-time-programmable (OTP) windowless packages. These devices are not erasable by the manufacturer or the user after assembly. Although each cell has probably been programmed and erased once in prior processing, the cells cannot be reprogrammed, tested, and subsequently erased after packaging. The E<sup>2</sup>CMOS cell based logic devices can. Alternatively, EPROM cell based logic device manufacturers rely upon a correlation to a "phantom array" for programmability and performance verification. This is the same scheme the bipolar manufacturers implement to test their programmable logic devices and is clearly inferior to the E<sup>2</sup>CMOS cell based approach.

The floating gate technology used in GAL devices is based on the well known Fowler-Nordheim Tunneling process. This process is based on a negligible current, controlled voltage programming process. The fast electrical erasure of the GAL device allows endurance to be guaranteed through actual test cycling as well as providing the opportunity to pattern the device many times during the manufacturing process to examine the actual functionality of the logic elements under direct control of the array cells exactly as the device would function under the control of a user defined pattern.

The E<sup>2</sup>CMOS cell is larger than a corresponding EPROM type cell. Historically this was a concern with PROM devices as they are die size limited by array cells; however, a programmable logic device is not as array intensive as the PROM. In fact, the GAL device array occupies only 6-7% of the total die area. The impact of the larger cell size on total die area is only 2-3%, a trivial amount to trade off for full testability by both manufacturer and user. This tradeoff definitely results in lower overall cost to the user since there no longer is a concern with (non-existent) reject devices.

GAL devices also incorporate a proprietary Margin Test feature that allows the actual charge potential of each cell to be verified individually to ensure full programming margins. To verify retention, all GAL devices are patterned and baked for 48 hours to be sure that the individual cells retain their data. Weak cells that lose some, but not all, of their charge would not normally be detectable without this analog bit-by-bit margin test feature. The user of Lattice devices can have full confidence in the twenty year minimum retention specification of these devices, again due to the actual test performed on the cell. Currently approved programming hardware verifies that the cells are programmed well beyond the minimum margins required for proper retention and performance.

The GAL family of devices guarantees through actual test the programmability of each cell, its ability to retain charge for the specified time period, and its ability to withstand repeated write/erase cycles (endurance). The

impact of this "actual test philosophy is directly observable in the quality levels of the GAL devices.

#### FUNCTIONAL VERIFICATION

GAL devices are tested under many different conditions during manufacture. To minimize the test time during some of the early (wafer probe) test operations where functionality of various portions of the circuit are being verified for the first time, it is to Lattice's advantage to test as quickly as possible. We have incorporated some special features into the GAL device which allow the array to be overridden during the proprietary Logic Test Mode. Data from the serial shift register overrides the array and travels through the sense amps so that the subsequent logic circuitry is driven by the actual signals it will see in normal operation. This test mode is more comprehensive than some of the alternate approaches used in previous design since the sense amp is not overridden. The sense amp is a critical untested portion of the circuitry in other PLDs. The Logic Test Mode allows complete testing and verification of the sense amps.

The functional testing of the output logic can be done in microseconds as opposed to milliseconds since it is not necessary to pattern the array to force a particular configuration or data condition. The cost savings of this technique over many test probes and insertions is dramatic. This manufacturing cost savings is passed along to the user in the form of higher quality devices with lower effective ASP devices.

The Logic Test Mode is not used to test the switching performance of the device. Actual array and architecture programming is necessary to verify the switching performance under normal conditions. Some 10 to 12 different array patternings are performed during each of the various configurations of the device under worst case patterning conditions. Devices using fuse-link technology must use rely on correlation and the EPROM based devices are restricted to 1 or 2 compromised patterns due to the lack of a cost/time effective erase capability.

## TESTING GAL DEVICES IN SYSTEM

The GAL family supports register preload to allow testing of sequential device configurations in a system or quickly on ATE. The current state can be quickly preloaded so that all of the various transitions to a next state can be tested. This feature is supported by the JEDEC standard data transfer protocol for test vector transfer. Many of the major device programmers also support the preload feature to functionally test a patterned PLD.

The implementation of preload in the GAL devices is somewhat unique in that it utilizes a serial shift scheme that allows both a load and unload of the

registers. The serial-in, serial-out shift technique allows the system designer or test engineer to devise a daisy-chaining scheme for forcing data. This scheme allows 100% controllability and observability of the device states.

## SUMMARY

The GAL20V8 and GAL16V8 bring a new level of flexibility and quality to the user of programmable logic. The generic architecture approach, E<sup>2</sup>CMOS, technology, and the philosophy of guaranteeing performance through actual test makes the GAL family of devices the ideal choice for both prototyping and volume production environments.

The functional testing of the output logic can be done in accordance as opposed to minimum delay it is not necessary to pattern the array to force a particular configuration or data condition. The cost savings of this technique over many test probes and inverters is dramatic. This manufacturing cost savings is passed along to the user in the form of higher quality devices with lower effective test delays.

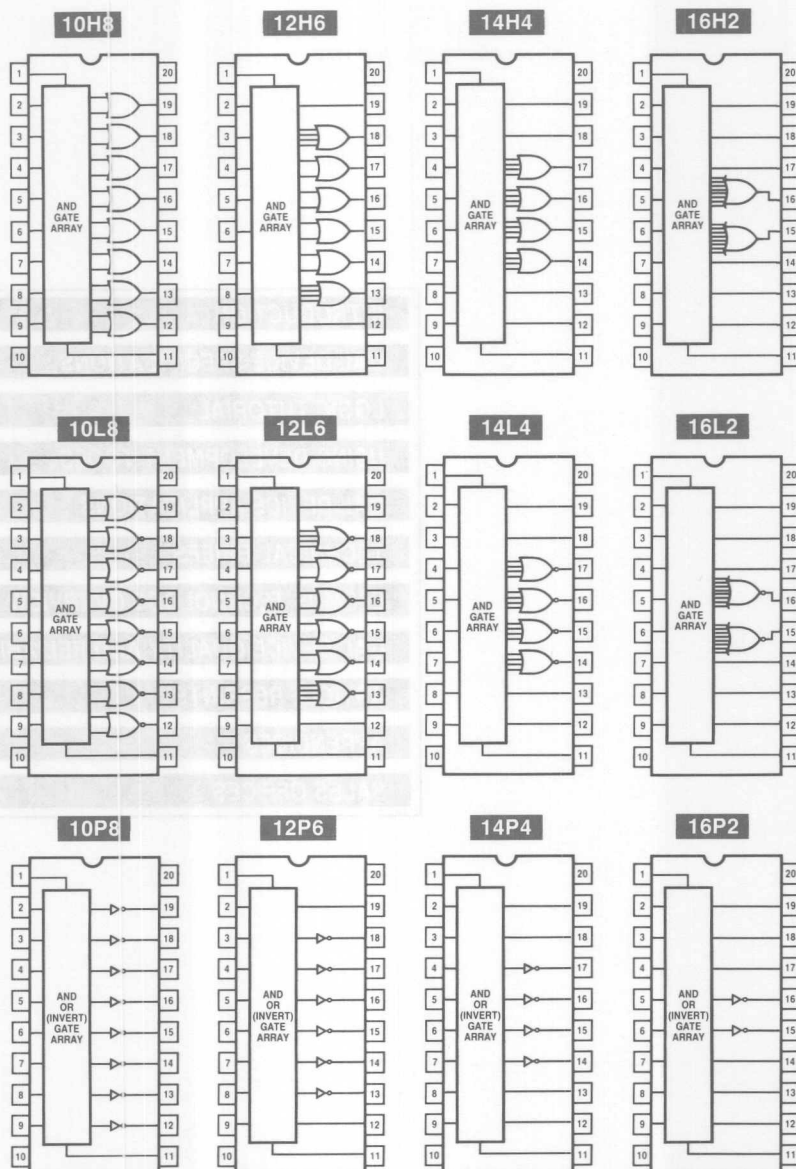
The logic test mode is not used to test the switched performance of the device. Actual array and architecture performance is necessary to verify the programmed performance under normal conditions. Some of the different array patterns are patterned during one of the various configurations of the device under worst case patterning conditions. Device using test-link technology does not rely on correlation and the E<sup>2</sup>CMOS based devices are restricted to 1 or 2 configurations because of the lack of a test-link alternative array capability.

GAL devices also incorporate a proprietary margin test feature that allows the actual change potential of each cell to be verified individually to ensure full programming margins. To ensure retention, all GAL devices are verified retention. All GAL devices are patterned and tested for 16 hours to be sure that the individual cells retain their data. Most cells that lose data, but not all, of their change would not normally be detectable without this margin bit-by-bit margin test feature. The user of Lattice devices can have full confidence in the twenty year minimum retention specification of these devices, again due to the actual test pattern on the cell. Currently approved programming patterns verified that the cells are programmed well beyond the minimum margins required for proper retention and performance.

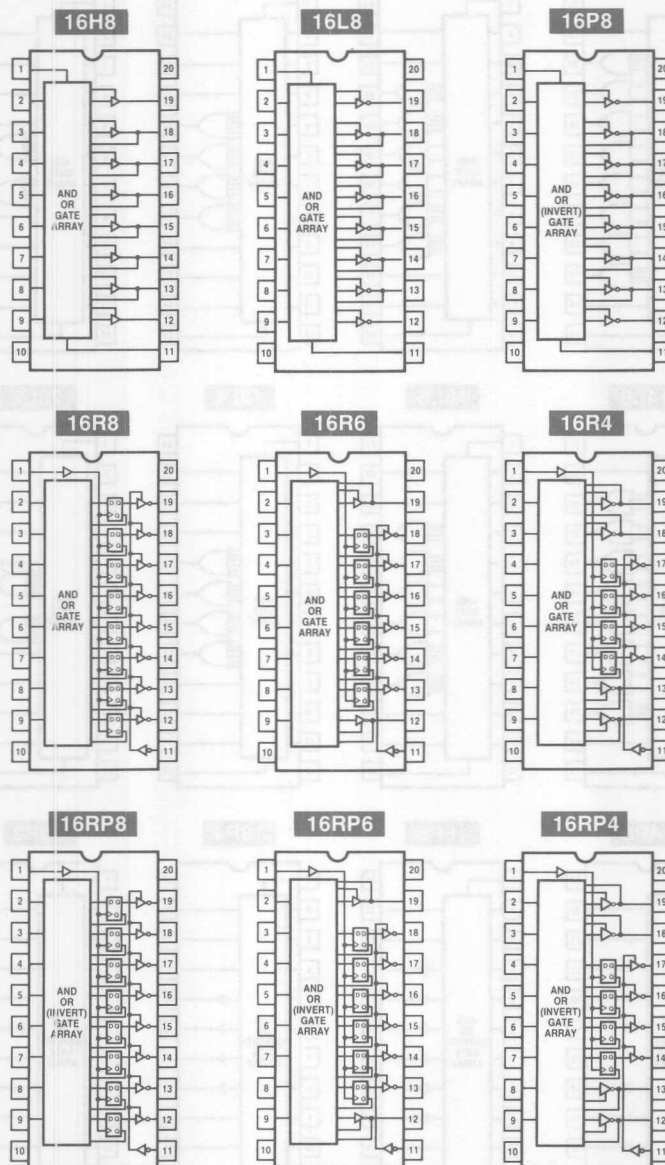
The GAL family of devices guarantees through actual test the programmability of each cell, the ability to retain change for the specified time period, and its ability to withstand repeated write/erase cycles (endurance). The



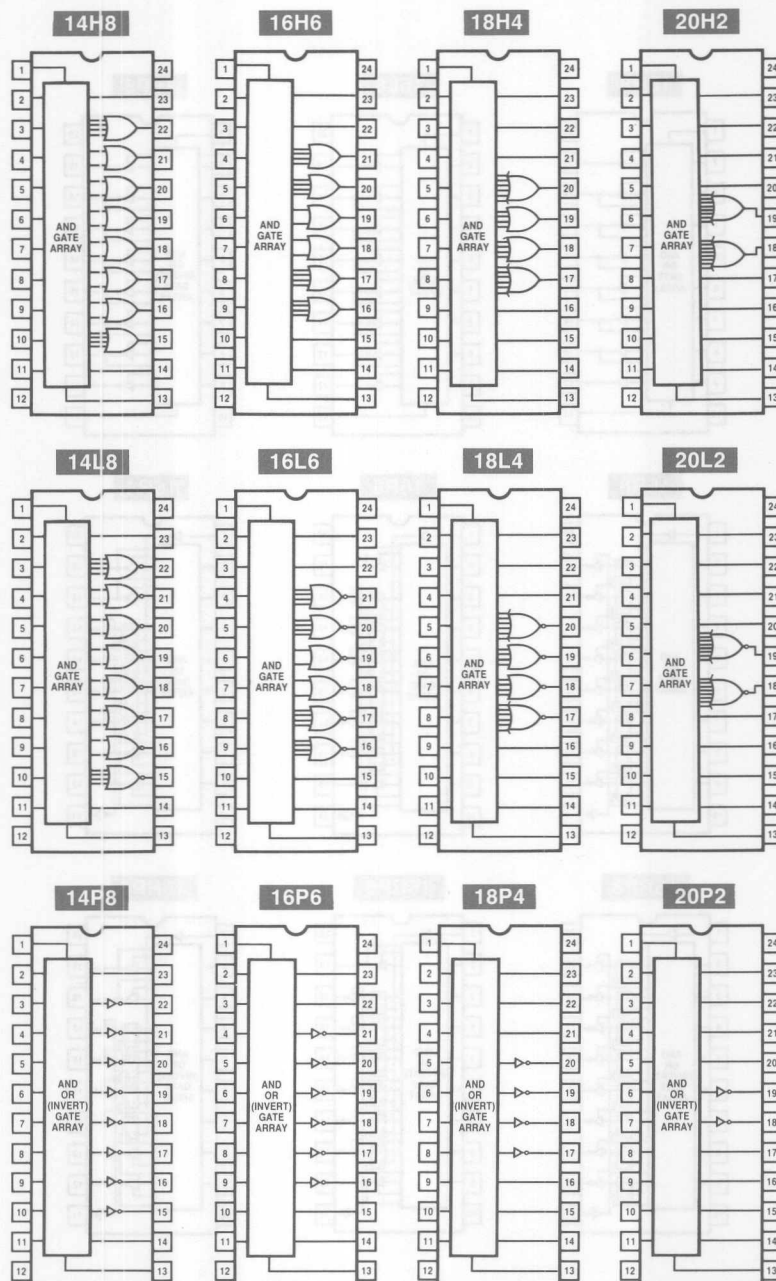




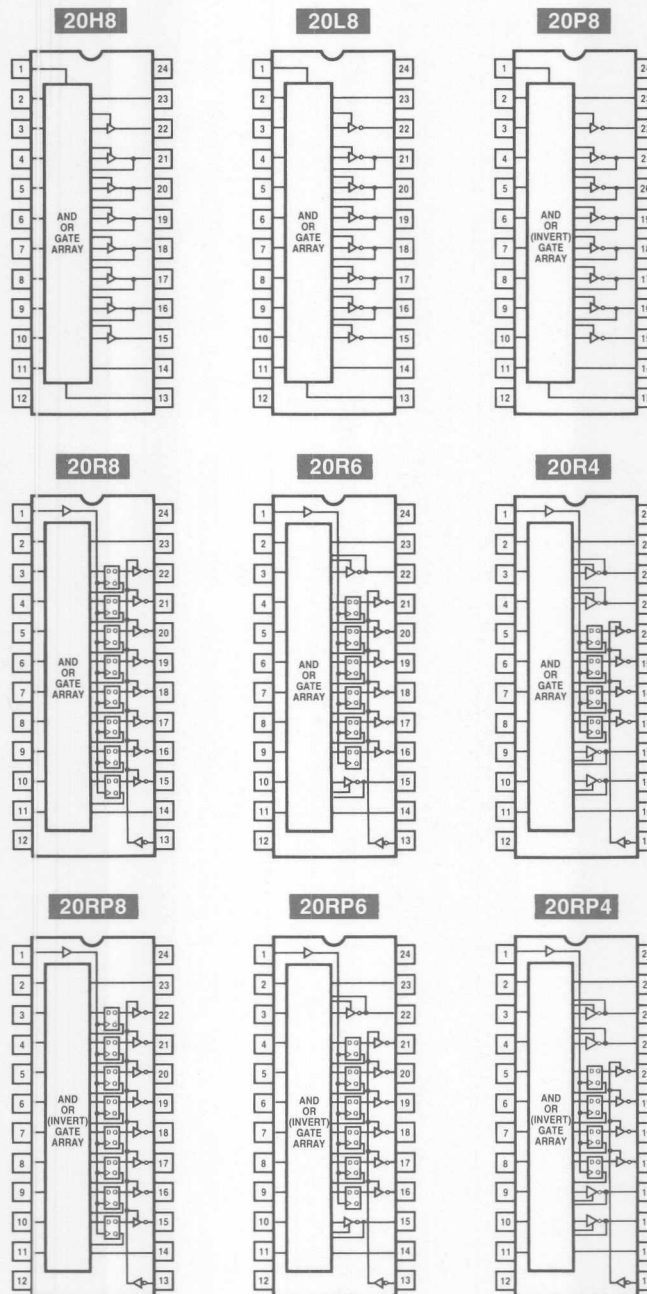
**Please Note:** The architectures represented above correspond to 'industry-standard' programmable logic devices. The GAL16V8 is by no means limited to these few. Please see Sections 3 and 6 for explanation of how GAL device Output Logic Macrocells (OLMCs) provide limitless architectural possibilities.



**Please Note:** The architectures represented above correspond to 'industry-standard' programmable logic devices. The GAL16V8 is by no means limited to these few. Please see Sections 3 and 6 for explanation of how GAL device Output Logic Macrocells (OLMCs) provide limitless architectural possibilities.

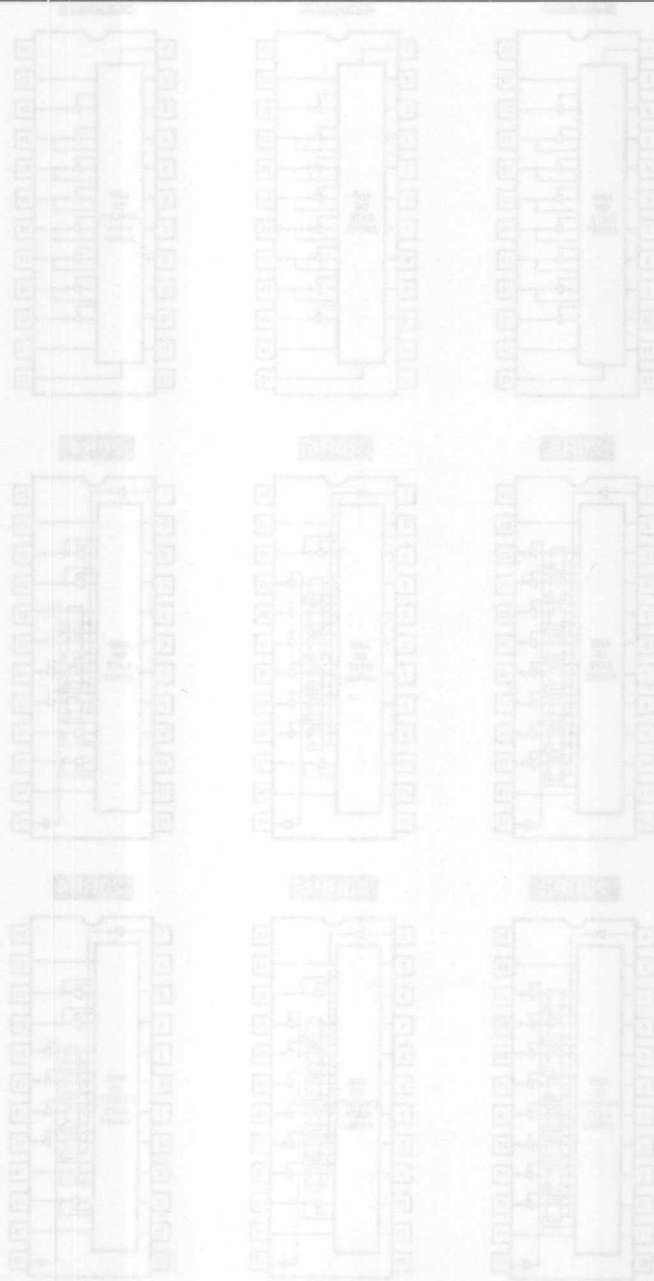


**Please Note:** The architectures represented above correspond to 'industry-standard' programmable logic devices. The GAL20V8 is by no means limited to these few. Please see Sections 3 and 6 for explanation of how GAL device Output Logic Macrocells (OLMCs) provide limitless architectural possibilities.



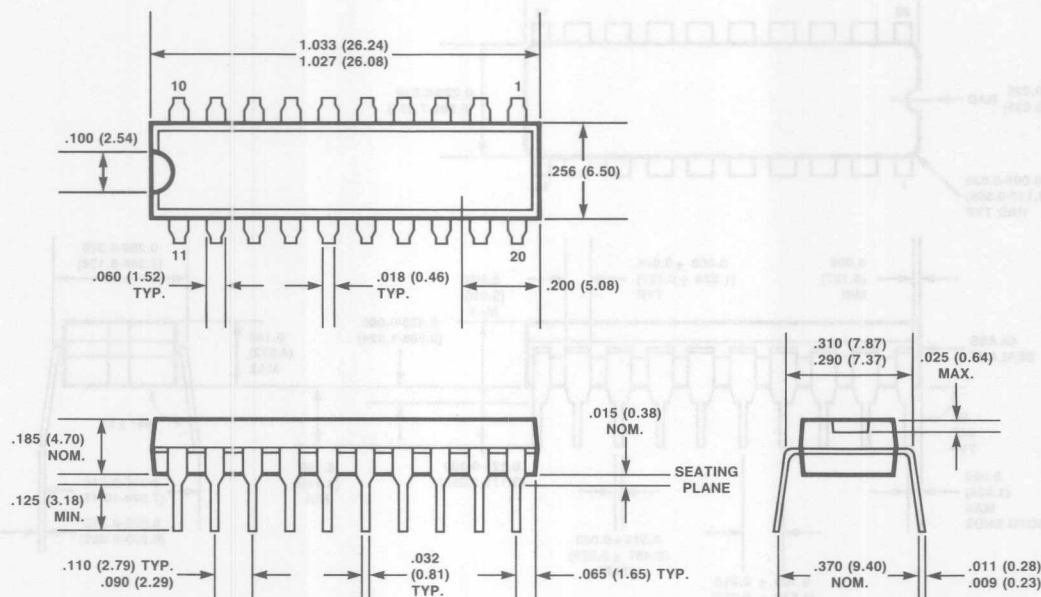
10

**Please Note:** The architectures represented above correspond to 'industry-standard' programmable logic devices. The GAL20V8 is by no means limited to these few. Please see Sections 3 and 6 for explanation of how GAL device Output Logic Macrocells (OLMCs) provide limitless architectural possibilities.

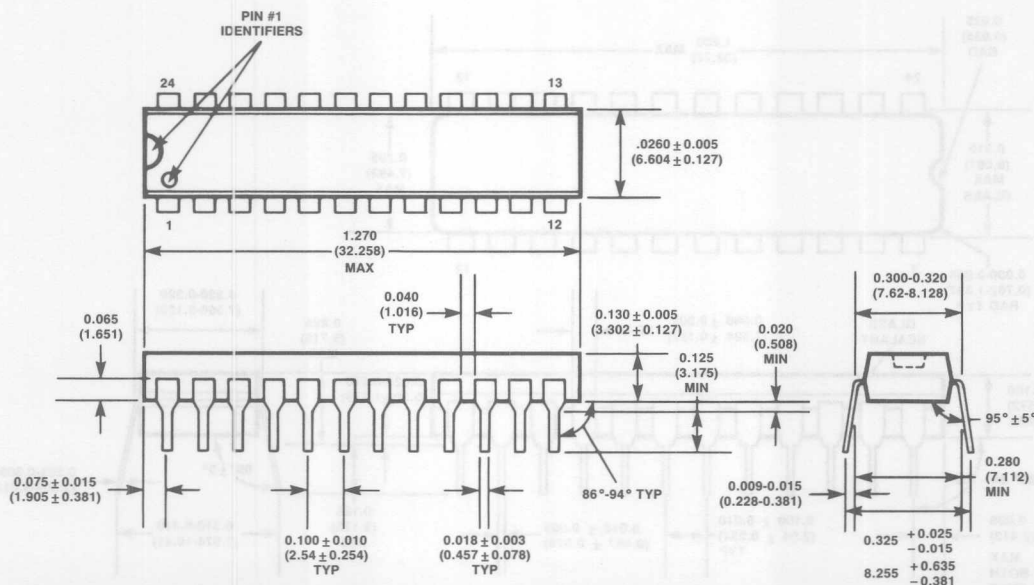


Pin 1 is the ground pin. Pin 2 is the VCC pin. Pin 3 is the VCC pin. Pin 4 is the VCC pin. Pin 5 is the VCC pin. Pin 6 is the VCC pin. Pin 7 is the VCC pin. Pin 8 is the VCC pin. Pin 9 is the VCC pin. Pin 10 is the VCC pin. Pin 11 is the VCC pin. Pin 12 is the VCC pin. Pin 13 is the VCC pin. Pin 14 is the VCC pin. Pin 15 is the VCC pin. Pin 16 is the VCC pin. Pin 17 is the VCC pin. Pin 18 is the VCC pin. Pin 19 is the VCC pin. Pin 20 is the VCC pin. Pin 21 is the VCC pin. Pin 22 is the VCC pin. Pin 23 is the VCC pin. Pin 24 is the VCC pin. Pin 25 is the VCC pin. Pin 26 is the VCC pin. Pin 27 is the VCC pin. Pin 28 is the VCC pin.

## 20-PIN, 300-MIL PLASTIC DIP

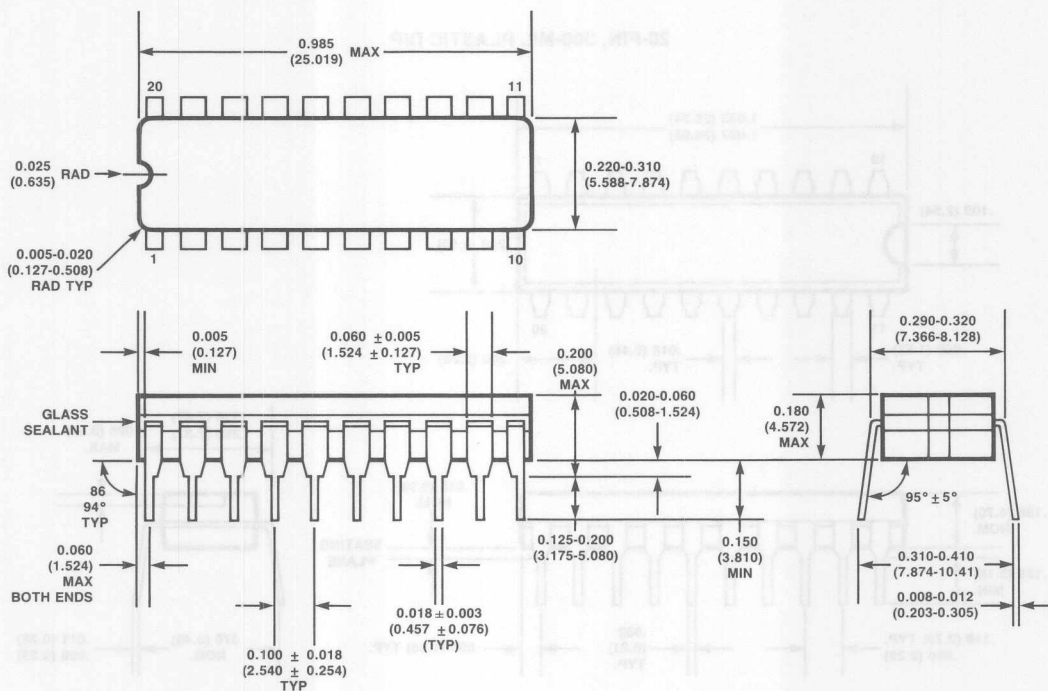


## 24-PIN, 300-MIL PLASTIC DIP

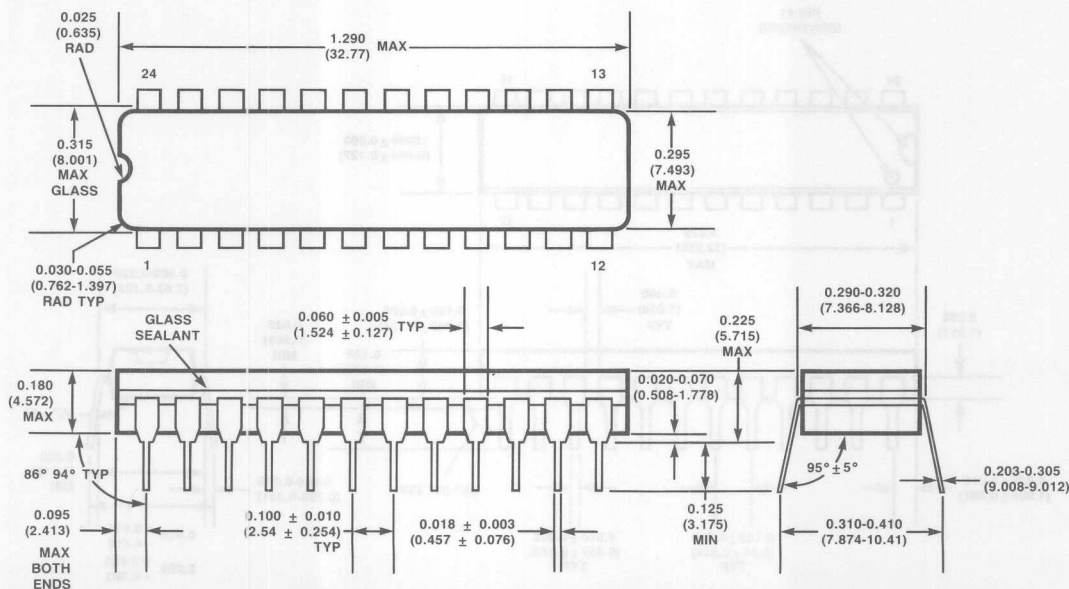


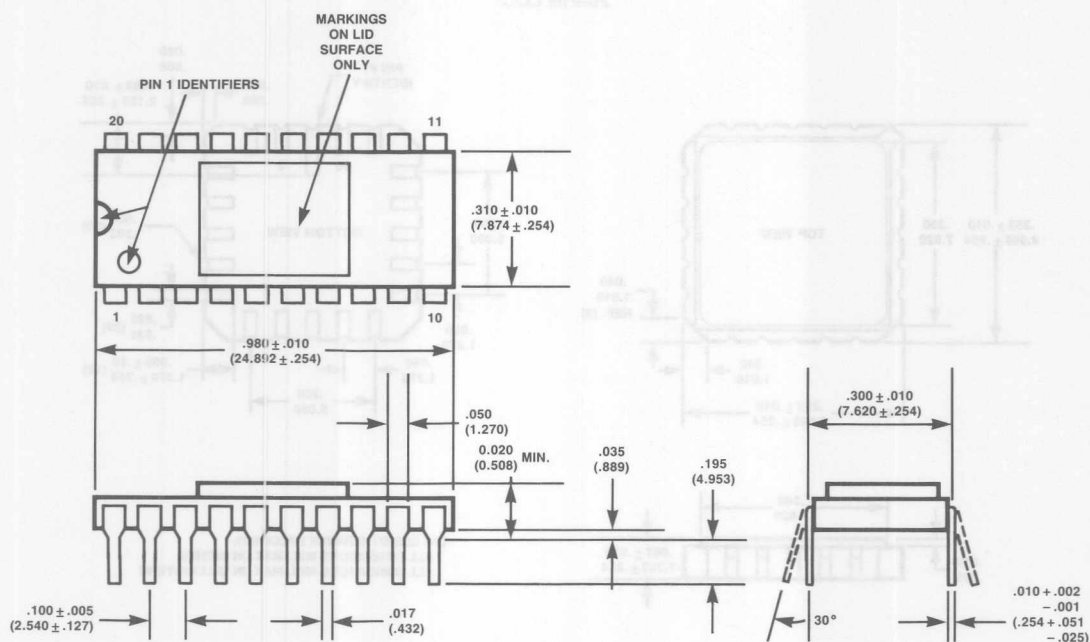
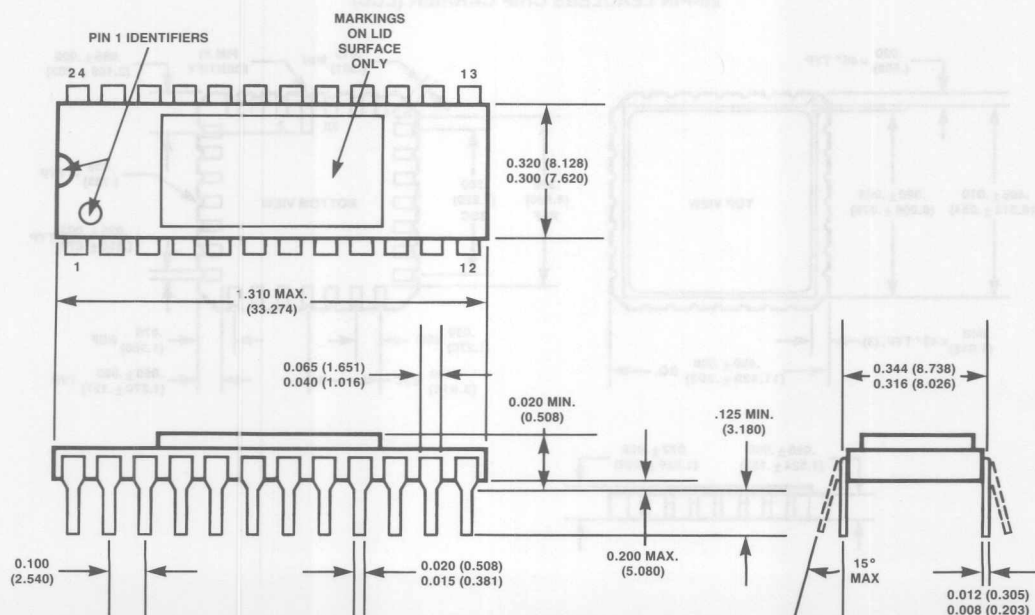


20-PIN, 300-MIL CERDIP

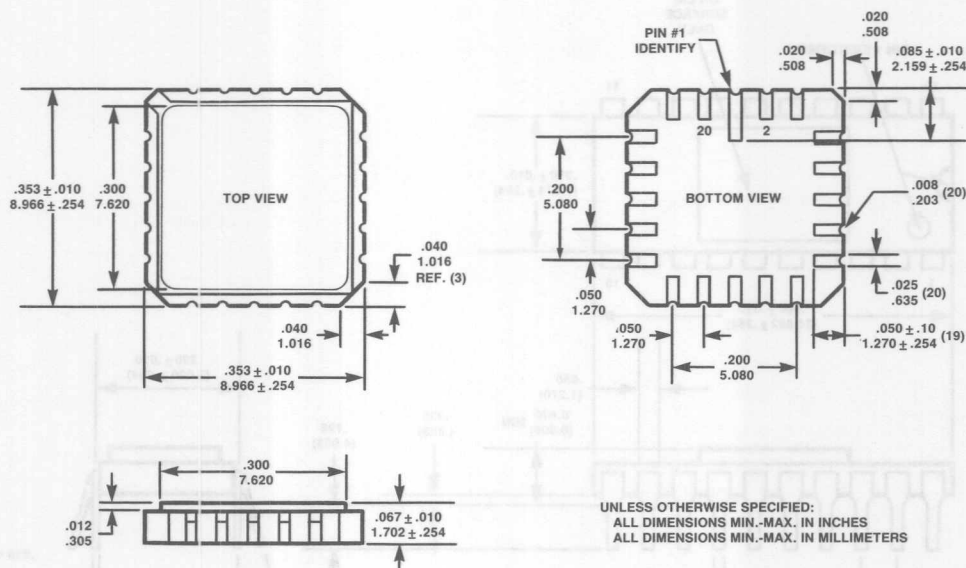


24-PIN, 300-MIL CERDIP

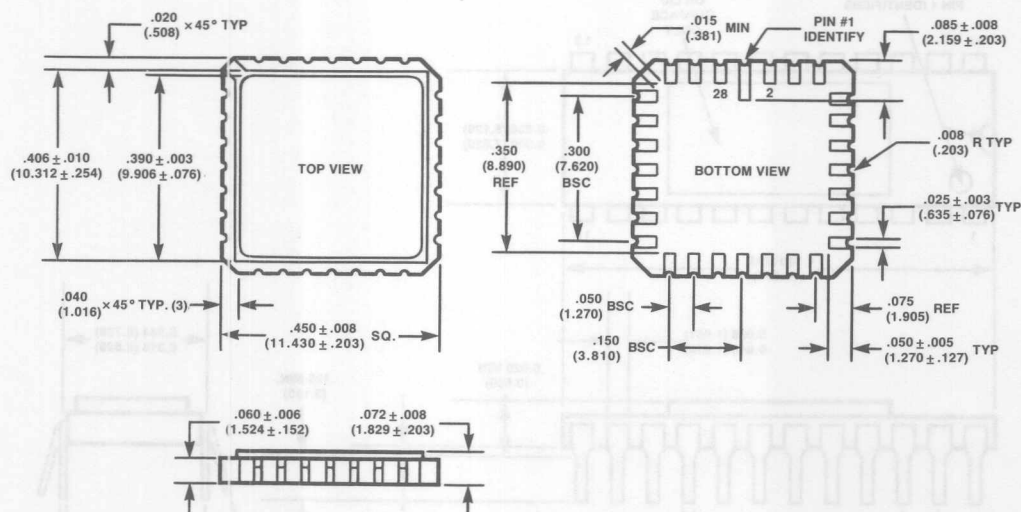


20-PIN, 300-MIL  
SIDE-BRAZED CERAMIC DIP

 24-PIN, 300-MIL  
SIDE-BRAZED CERAMIC DIP


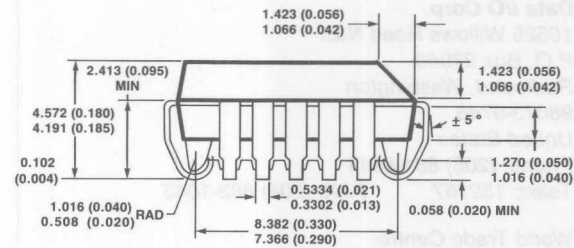
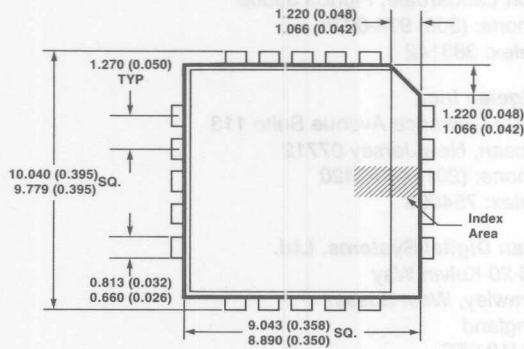
## 20-PIN LCC



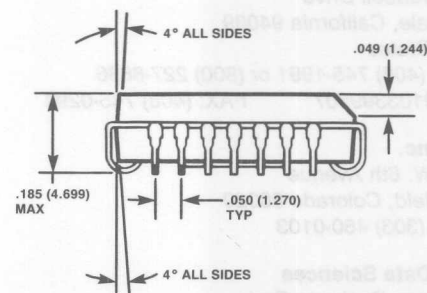
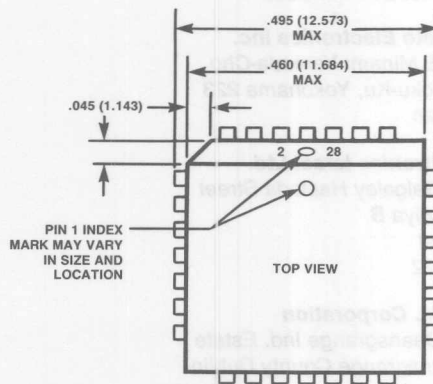
### 28-PIN LEADLESS CHIP CARRIER (LCC)



20-PIN PLCC



28-PIN PLASTIC LEADED CHIP CARRIER PLCC



All GAL devices manufactured by Lattice Semiconductor are supported by third-party development tools. The manufacturers of these tools are listed below for your reference. Contact Lattice for a current list of qualified suppliers and revisions.

GAL devices can be programmed by your local Lattice authorized distributor as well. ☐

## PROGRAMMER MANUFACTURERS

### Data I/O Corp.

10525 Willows Road N.E.  
P.O. Box 97046  
Redmond, Washington  
98073-9746  
United States  
Phone: (206) 881-6444  
Telex: 152167 FAX: (206) 882-1043

World Trade Centre  
Strawinskylaan 633  
1107 XX Amsterdam  
The Netherlands  
Phone: (20) 622866  
Telex: 4740166 FAX: (20) 627255

### Stag Electronic Designs

Tewin Court  
Welwyn Garden City  
Hertfordshire AL7 1AU  
United Kingdom  
Phone: (07073) 32148  
FAX: (07073) 71503  
Telex: 8953451

528-5 Weddell Drive  
Sunnyvale, California 94089  
United States  
Phone: (408) 745-1991 or (800) 227-8836  
Telex: 9103399607 FAX: (408) 745-0294

### Inlab, Inc.

2150-I W. 6th Avenue  
Broomfield, Colorado 80020  
Phone: (303) 460-0103

### Valley Data Sciences

Charleston Business Park  
2426 Charleston Road  
Mountain View, California 94043  
Phone: (415) 968-2900  
Telex: 4993461

### Varix

1210 E. Campbell Road Suite 100  
Richardson, Texas 75081  
Phone: (214) 437-0777  
Telex: 203906

### Oliver Advanced Engineering (OAE)

320 W. Arden Avenue Suite 220  
Glendale, California 91203  
Phone: (818) 240-0080  
Telex: 62914399

### Logical Devices

1321 North West 65th Place  
Fort Lauderdale, Florida 33309  
Phone: (305) 974-0975  
Telex: 383142

### Digelec Inc.

1602 Lawrence Avenue Suite 113  
Ocean, New Jersey 07712  
Phone: (201) 493-2420  
Telex: 754098

### Elan Digital Systems, Ltd.

16-20 Kelvin Way  
Crawley, West Sussex  
England  
RH10 2TS  
Telex: 877314 FAX: (02935) 18591

### Elan Digital Systems (ESP)

P.O. Box 1610  
San Anselmo, California 94960  
Phone: (408) 734-2226

### Japan Macnics Corp. (JMC)

516 Imaiminami-Cho, Nakahara-ku  
Kawasaki-City, 211  
Japan  
Phone: 044-711-0022

### Minato Electronics Inc.

4105 Minami Yamada-Cho  
Kohoku-Ku, Yokohama 223  
Japan

### Digitronics Israel Ltd.

25 Galgaley Haplada Street  
Herzliya B  
Israel  
46722

### AVAL Corporation

11 Deansgrange Ind. Estate  
Deansgrange County Dublin  
Ireland  
Phone: (01) 850533  
Telex: 90144 FAX: (01) 850031

### Sunrise Electronics

524 South Vermont Avenue  
Glendora, California 91740  
Phone: (818) 914-1926  
Telex: 5106011165

**Digital Media**

3178 Gibraltar Avenue  
Costa Mesa, California 92626  
Phone: (714) 751-1373

**Micropross**

5 Rue Denis-Papin  
Parc d'activités des prés  
59650 Villeneuve-d'Ascq  
France  
Phone: 20-47-90-40  
Telex: 120611

**Stack**

Unit 8  
Wedgewood Road  
Bicester, Oxon  
OX6 7UL  
United Kingdom  
Phone: 44-869-240404

**Kontron Electronics**

1230 Charleston Road  
Mountain View, California 94039-7230

**PROGRAMMING INFORMATION**

GAL devices should be programmed using data formatted within the JEDEC-recommended standard for PLD object code.

Copies of JEDEC Standard No. 3 (JC-42.1) may be obtained from Lattice Semiconductor or by writing to the following address:

JEDEC Executive Secretary  
**Electronics Industries Association**  
2001 Eye Street N.W.  
Washington, D.C. 20006  
United States

**SOFTWARE DEVELOPERS****Data I/O / FutureNet**

10525 Willows Road N.E.  
P.O. Box 97046  
Redmond, Washington 98073-9746  
United States  
Phone: (206) 881-6444  
Telex: 152167 FAX: (206) 882-1043

**Assisted Technology / Personal CAD**

1290 Parkmoor Avenue  
San Jose, California 95126  
Phone: (408) 971-1300  
Telex: 882439 FAX: (408) 279-3752

**Valley Data Sciences**

Charleston Business Park  
2426 Charleston Road  
Mountain View, California 94043  
Phone: (415) 968-2900  
Telex: 4993461

**IDATA GmbH**

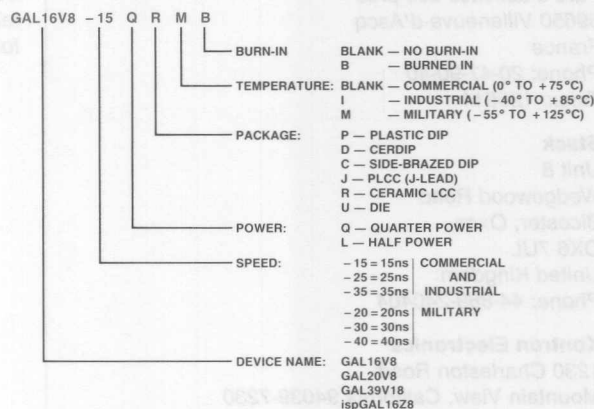
Hansastraße 29A  
D-7500 Karlsruhe 21  
West Germany  
Phone: 0721-57-9509

**NOTE:** Lattice Semiconductor assumes no responsibility for the suitability or accuracy of third-party development tools. Lattice will provide, upon request, a list of qualified and approved suppliers indicating a factory qualification of the tool has been performed.



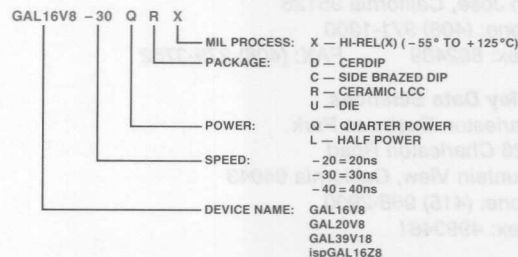
## STANDARD PRODUCT

- ☐ Electrically tested per LATTICE Data Sheet.
- ☐ Supplied in hermetic and molded packages.
- ☐ Commercial, Industrial, or Military temperature range.
- ☐ Additional burn-in available.



## HI-REL(X) PRODUCT

- ☐ Screened per MIL-STD-883C, Method 5004.
- ☐ Includes 160-hour burn-in at 125°C.
- ☐ Electrically tested per LATTICE Data Sheet.
- ☐ Supplied in hermetic package only.
- ☐ Military temperature range only.



## MIL-STD-883C PRODUCT

- ☐ Screened per MIL-STD-883C Method 5004, Class B.
- ☐ Includes 160-hour burn-in at 125°C
- ☐ Quality conformance testing, Method 5005, Class B, Groups A, B, C, and D.
- ☐ AC at 25°C, d.c. and functional testing at 25°C as well as temperature and power supply extremes performed on 100% of every lot.
- ☐ Supplied in hermetic package only.
- ☐ Military temperature range only.

GAL16V8 -30 Q R /883C

MIL PROCESS: /883C — FULL 883C CONFORMANCE

PACKAGE: D — CERDIP  
C — SIDE-BRAZED DIP  
R — CERAMIC LCC  
U — DIE

POWER: Q — QUARTER POWER  
L — HALF POWER

SPEED: — 20 = 20ns  
— 30 = 30ns  
— 40 = 40ns

DEVICE NAME: GAL16V8  
GAL20V8  
GAL39V18  
IspGAL16Z8

## SPEED/POWER CROSS-REFERENCE GUIDE

SPEED	POWER	GAL DEVICE	BIPOLAR PAL DEVICE
15ns	45mA	— 15Q	—
15ns	90mA	— 15L	—
15ns	180mA	use — 15L	B
* 20ns	50mA	— 20Q	—
* 20ns	90mA	— 20L	—
* 20ns	210mA	use — 20L	B MIL
25ns	45mA	— 25Q	—
25ns	90mA	— 25L	B-2
25ns	180mA	use — 25L	A
* 30ns	50mA	— 30Q	—
* 30ns	90mA	— 30L	B-2 MIL
* 30ns	210mA	use — 30L	A MIL
35ns	45mA	— 35Q	B-4
35ns	90mA	— 35L	A-2
35ns	180mA	use — 35L	STD
* 40ns	50mA	— 40Q	B-4 MIL
* 40ns	90mA	— 40L	A-2 MIL
* 40ns	210mA	use — 40L	STD MIL

\* MILITARY TEMPERATURE RANGE

